US006233683B1

(12) **United States Patent**
Chan et al.

(10) **Patent No.: US 6,233,683 B1**
(45) **Date of Patent: May 15, 2001**

(54) **SYSTEM AND METHOD FOR A MULTI-APPLICATION SMART CARD WHICH CAN FACILITATE A POST-ISSUANCE DOWNLOAD OF AN APPLICATION ONTO THE SMART CARD**

(75) Inventors: **Alfred Chan**, Daly City; **Marc B. Kekicheff**, Palo Alto; **Joel M. Weise**, Burlingame; **David C. Wentker**, San Francisco, all of CA (US)

(73) Assignee: **Visa International Service Association**, Foster City, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/046,994**

(22) Filed: **Mar. 24, 1998**

Related U.S. Application Data

(60) Provisional application No. 60/061,763, filed on Oct. 14, 1997, and provisional application No. 60/041,468, filed on Mar. 24, 1997.

(51) **Int. Cl.**[7] **G06F 1/24**
(52) **U.S. Cl.** **713/172**; 713/200; 713/201
(58) **Field of Search** 380/172; 713/172, 713/200, 201; 235/380

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,742,215	5/1988	Daughters et al. .	
4,831,245	5/1989	Igasawara .	
5,332,889	7/1994	Lundstrom et al. .	
5,378,884	1/1995	Lundstrom et al. .	
5,530,232	6/1996	Taylor .	
5,583,933	12/1996	Mark .	
5,715,431	* 2/1998	Everett et al.	235/492
5,901,303	* 5/1999	Chew	235/492

FOREIGN PATENT DOCUMENTS

100227	11/1994	(AT) .
19607363 A1	9/1996	(DE) .
0193635 A1	9/1986	(EP) .
0658862	6/1995	(EP) .
0795844	9/1997	(EP) .
0798673	10/1997	(EP) .
98/43212	10/1998	(WO) .

OTHER PUBLICATIONS

Carol Hovenga Fancher, "In Your Pocket SmartCard", Feb. 1997, IEEE Spectrum.

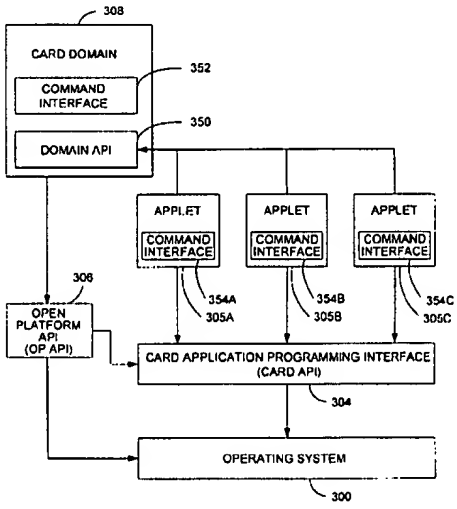
(List continued on next page.)

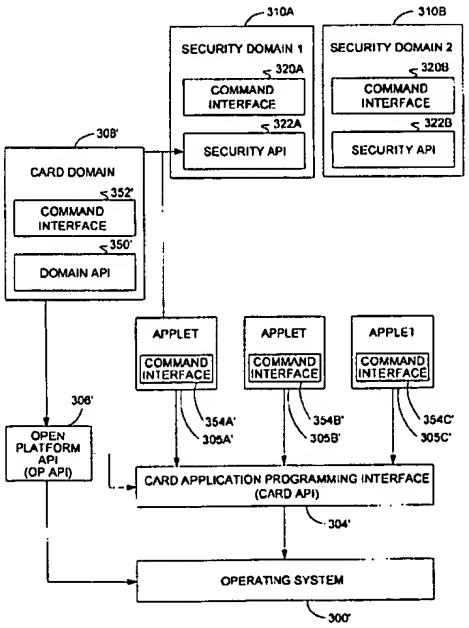
Primary Examiner—Thomas R. Peeso
(74) *Attorney, Agent, or Firm*—Beyer Weaver & Thomas, LLP

(57) **ABSTRACT**

The embodiments of the present invention teaches a system and method which allows card issuers to securely add applications during the lifetime of the card after the card has already been issued (post issuance). The system and method according to embodiments of the present invention allows the loading of an application and/or objects from an application server via a card acceptance device and its supporting system infrastructure delivery mechanism, onto a card post issuance in a secure and confidential manner.

23 Claims, 15 Drawing Sheets





OTHER PUBLICATIONS

Chaum et al., "SmartCard 2000: The Future of IC Cards", Oct. 19, 1987, Elsevier Science Publishers, B.V.
Steven Levy, "E-Money (That's What I Want)", Dec. 1994, Wired Magazine.
Carol H. Fancher, "Smart Cards as Potetial Applications Grow, Computers in the Wallet are Making Unobstrusive Inroads", Aug. 1996, Scientific American Website.
Jerome Svigals, "SmartCards The New Bank Cards", 1985, MacMillan Publishing Company.
Roy Bright, "SmartCards: Principles, Practice, Applications", 1998, Ellis Horwood Limited.
Jerome Svigals, "SmartCards The Ultimate Personal Computer", 1985, MacMillan Publishing Company.
Hawkes et al., "Integrated Circuit Cards, Tags and Tokens", 1990, BSP Professional Books.
Hiro Shogase, The Very Smart Card: A Plastic Packet Bank:, Oct. 1988, IEEE Spectrum.
David Naccache, "Cryptographic Smart Cards", Jun. 3, 1996, IEEE Micro 1966 Website.
Zoreda et al., "Smart Cards", 1994, Artech House.
"Identification Card Systems—Inter-Sector Electronic Purse Part I: Concepts and Structures", 1994, European Standard, PrEN 1546.
"Identification Card Systems—Intr-Sector Electronic Purse Part 2: Security Architecture", 1994, European Standard, prEN XXXXX-2.
"Identification Card System—Inter-Sector Electronic Purse Part 3: Data Elements and Interchanges", 1994, European Prestandard,. prEN 1546-3.
"Identification Card System—Inter-Sector Electronic Purse Part 4: Devices", 1994, European Prestandard, prEN 1546-4.
"Identification Cards—Integrated Circuit(s) Cards With Contacts Part 1: Physical Characteristics", 1987, International Standard, ISO 7816-1, First Edition.
"Identification Cards—Integrated Circuit(s) Cards With Contacts Part 2: Dimensions and Location of the Contacts", 1988, International Standard ISO 7816-2, First Edition.
"Identification Cards—Integrated Circuit(s) Cards With Contacts Part 3: Electronic Signals and Transmission Protocols", International Standard, ISO/IEC 7816-3, First Edition.
"Identification Cards—Integrated Circuit(s) Cards with Contacts Part 4: Inter-Industry Commands for Interchange", International Standard, ISO/IEC 7816-4, First Edition.
"Identification Cards—Integrated Circuit(s) Cards With Contacts Part 5: Numbering System and Registration Procedure for Application Identifiers", 1993, International Standard, ISO/IEC DIS 7816-5.
"International Cards—Integrated Circuit(s) Cards With Contacts Part 6: Inter-Industry Data Elements", 1995, International Standard, ISO/IEC DIS 7816-6.
"Bank Cards—Magnetic Strip Data Content For Track 3", 1987, International Standard, ISO 4909 Second Edition.
"Identification Cards—Physical Characteristics", 1995, International Standard, ISO/IEC 7810, Second Edition.
"Identification Cards—Recording Technique—Part 1: Embossing", 1995. International Standard, ISO/IEC 7811-1, Second Edition.
"Identification Cards—Recording Technique—Part 2: Magnetic Strip", 1995, International Standard, ISO/IEC 7811-2, Second Edition.
"Identification Cards—Recording Technique—Part 3: Location of Embossed Characters on ID-1 Cards", 1995, International Standard, ISO.IEC 7811-5, Second Edition.
"Identification Cards—Recording Technique—Part 4: Location of Read-Only Magnetic Tracks—tracks 1 & 2", 1995 International Standard, ISO/IEC 7811-4, Second Edition.
"Identification Cards—Recording Technique—Part 5: Location of Read-Write Magnetic Track—Trck 3", 21995, International Standard ISO.IEC 7811-5, Second Edition.
"Identification Cards—Recording Technique—Part 6: Magnetic Stripe-High Coercivity", 1996, International Standard, ISO/IEC 7811-6, First Edition.
"Identification Cards—Financial Transaction Cards", 1990, International Standard, ISO/IEC 7813, Third Edition.
"Identification Cards—Financial Transaction Cards Amendment 1" 1996, International Standard, ISO/IEC 7813, Fourth Eiditon.
"Identification Cards—Contactless Integrated Circuit(s) Cards—Part 1: Physical Characteristics", 1992, International Standard, ISO/IEC 10536-1, First Edition.
"Identification Cards—Contactless Integrated Circuit(s) Cards—Part 2: Dimensions and Location of Coupling Aresa", 1995, International Standard, ISO/IEC 10536-2, First Edition.
"Identification Cards—Contactless Integrated Circuit(s) Cards—Part 3: Electronic Signals and Reset Procedures", 1996, International Standard, ISO/IEC 10536-3, First Edition.
"Financial transaction Cards-Security Architecture of Financial Transaction System Using Integrated Circuit Cards- Part 1: Card Life Cycle", Sep. 15, 1991, International Standard, ISO 1020-1, First Edition.

* cited by examiner

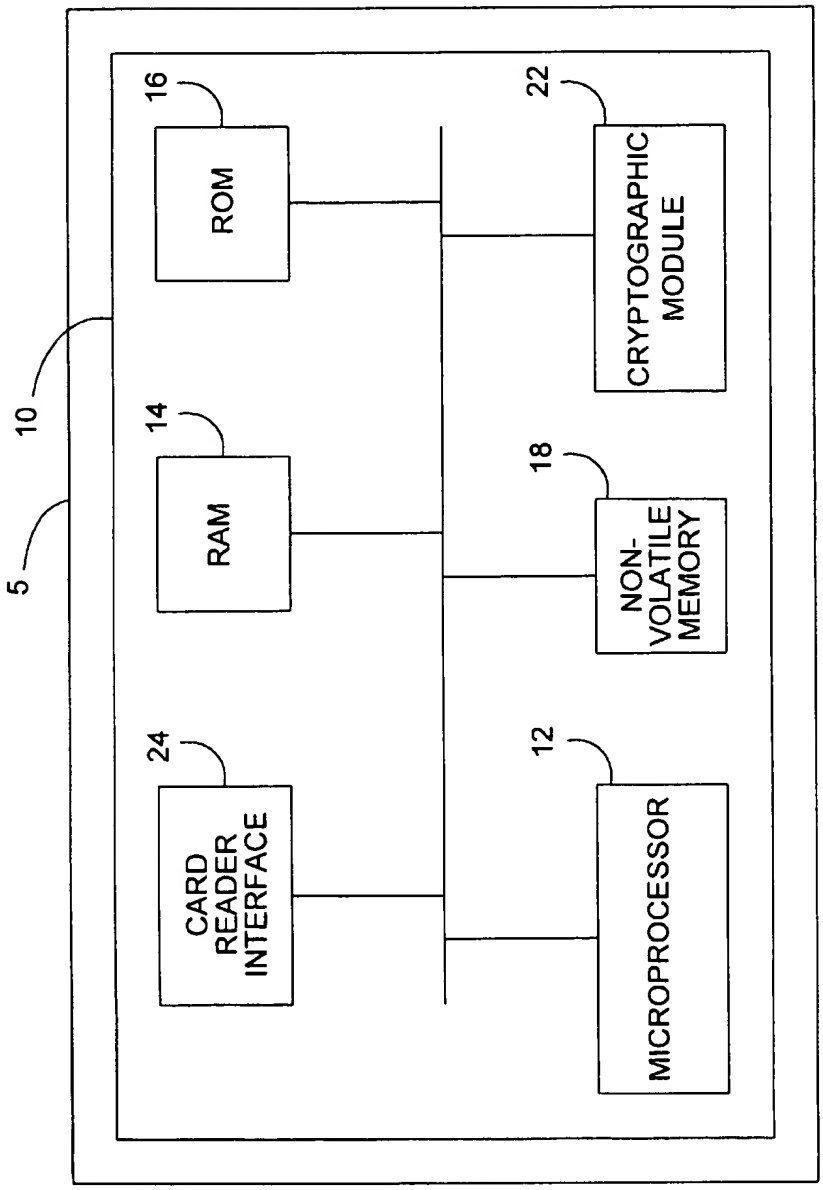


FIG. 1
(PRIOR ART)

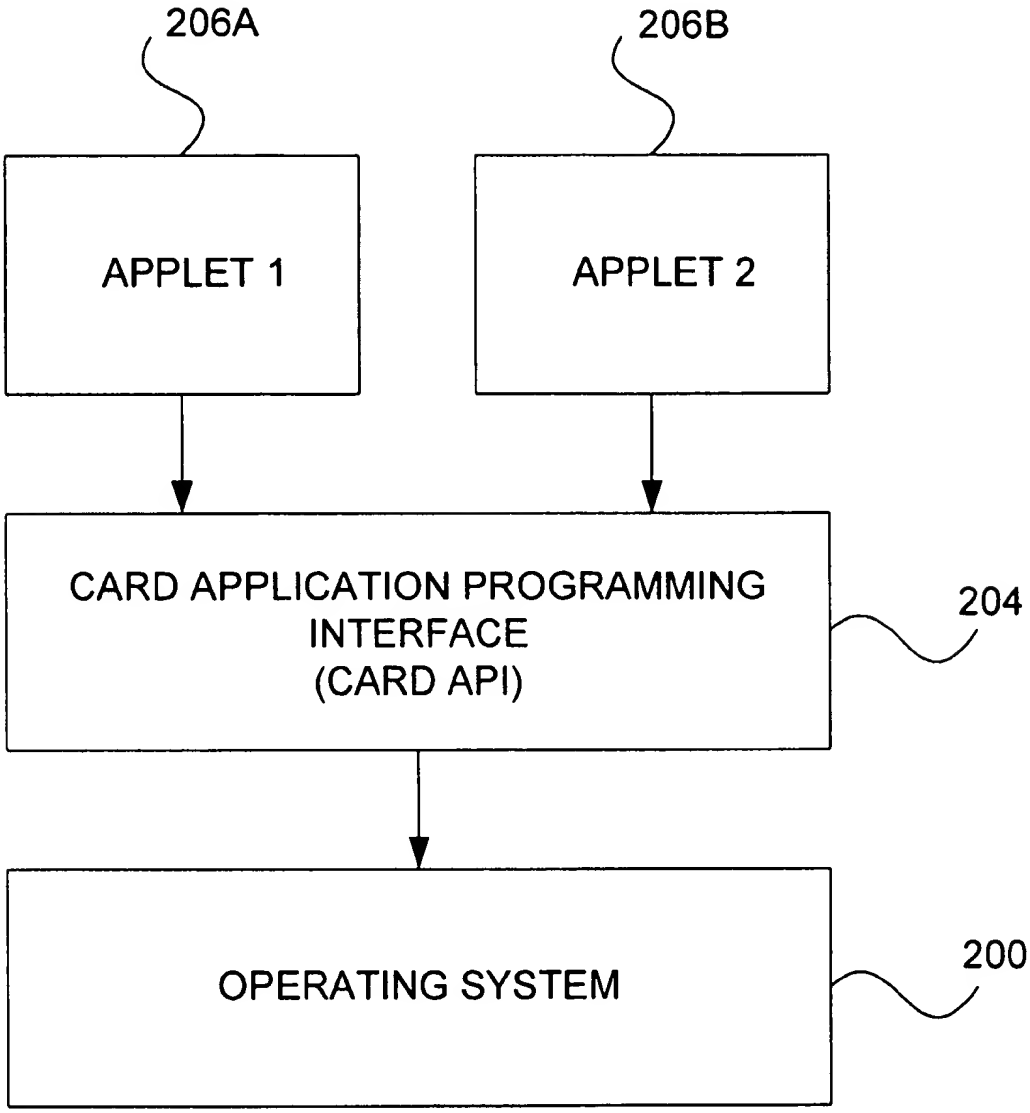


FIG. 2
(PRIOR ART)

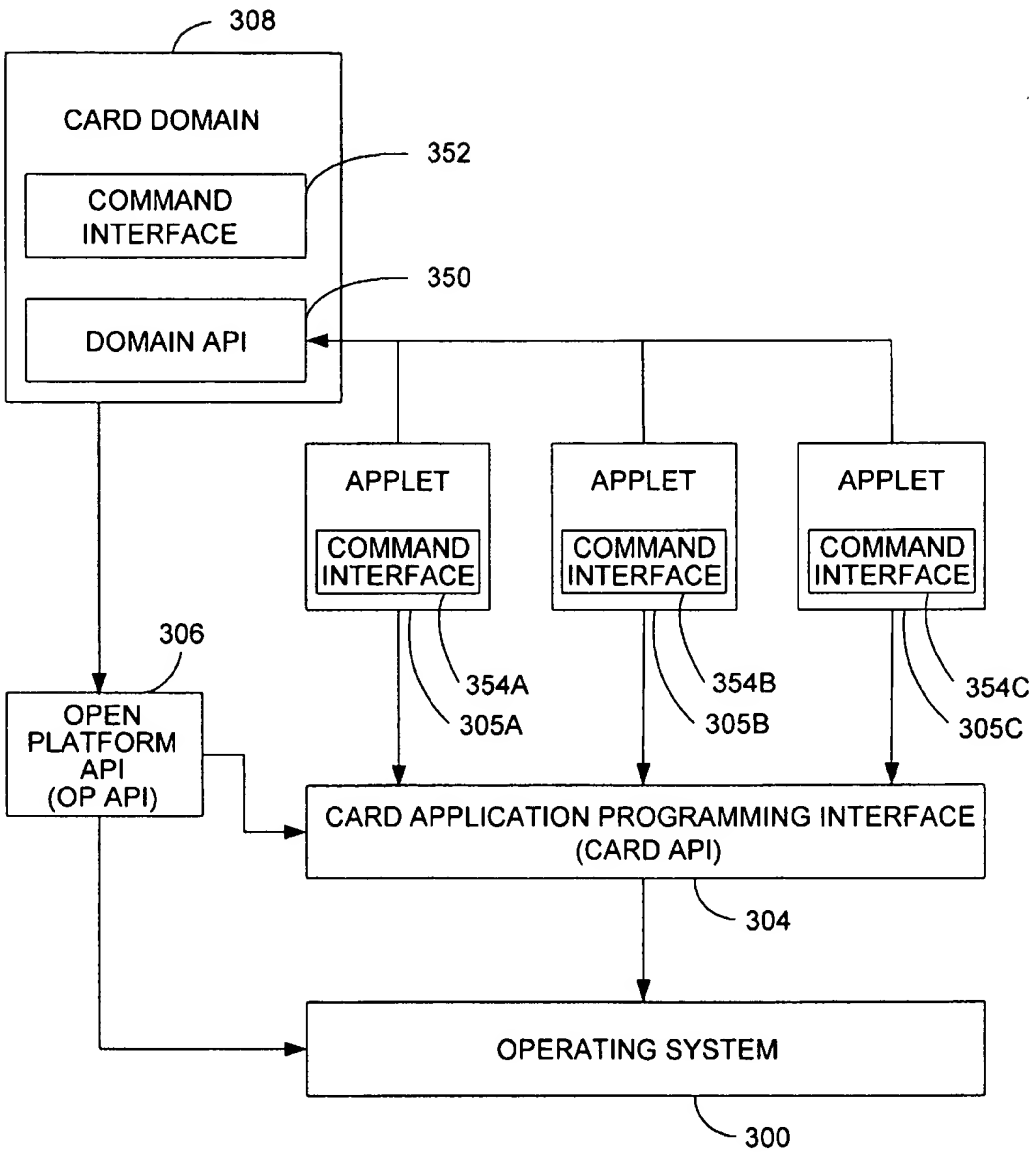

308
CARD DOMAIN
352
COMMAND INTERFACE
350
DOMAIN API
APPLET
COMMAND INTERFACE
APPLET
COMMAND INTERFACE
APPLET
COMMAND INTERFACE
306
OPEN PLATFORM API (OP API)
354A
305A
354B
305B
354C
305C
CARD APPLICATION PROGRAMMING INTERFACE (CARD API)
304
OPERATING SYSTEM
300

FIG. 3A

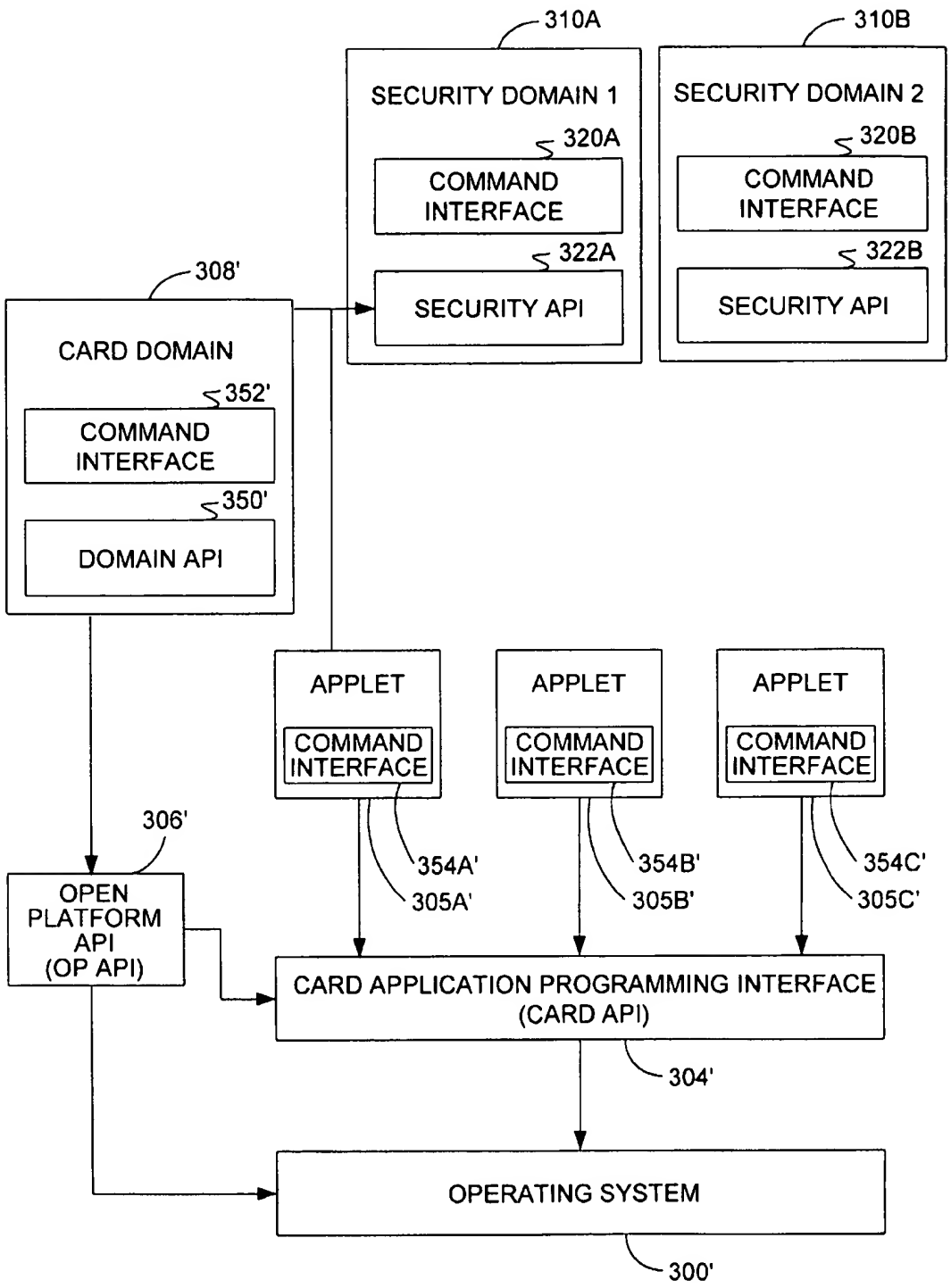

310A
SECURITY DOMAIN 1
320A
COMMAND INTERFACE
322A
SECURITY API
310B
SECURITY DOMAIN 2
320B
COMMAND INTERFACE
322B
SECURITY API
308'
CARD DOMAIN
352'
COMMAND INTERFACE
350'
DOMAIN API
APPLET
COMMAND INTERFACE
APPLET
COMMAND INTERFACE
APPLET
COMMAND INTERFACE
306'
OPEN PLATFORM API (OP API)
354A'
305A'
354B'
305B'
354C'
305C'
CARD APPLICATION PROGRAMMING INTERFACE (CARD API)
304'
OPERATING SYSTEM
300'

FIG. 3B

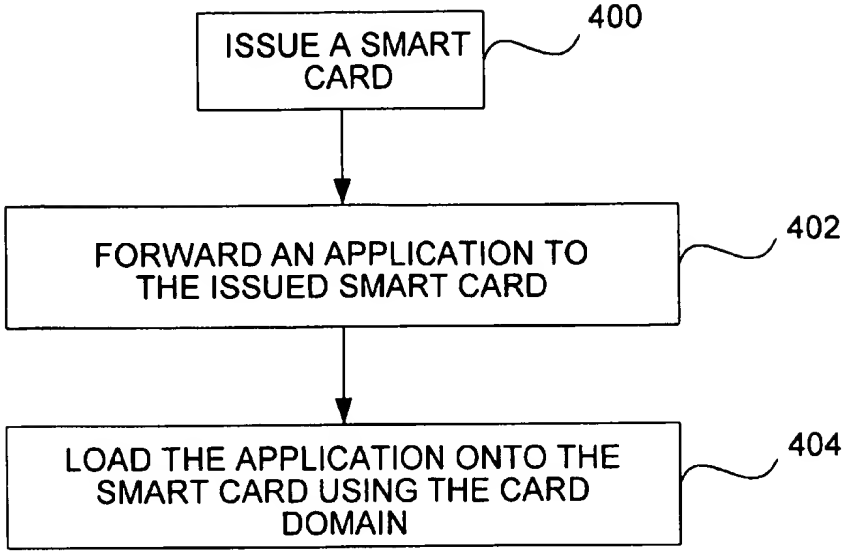

ISSUE A SMART CARD
400
FORWARD AN APPLICATION TO THE ISSUED SMART CARD
402
LOAD THE APPLICATION ONTO THE SMART CARD USING THE CARD DOMAIN
404

FIG. 4

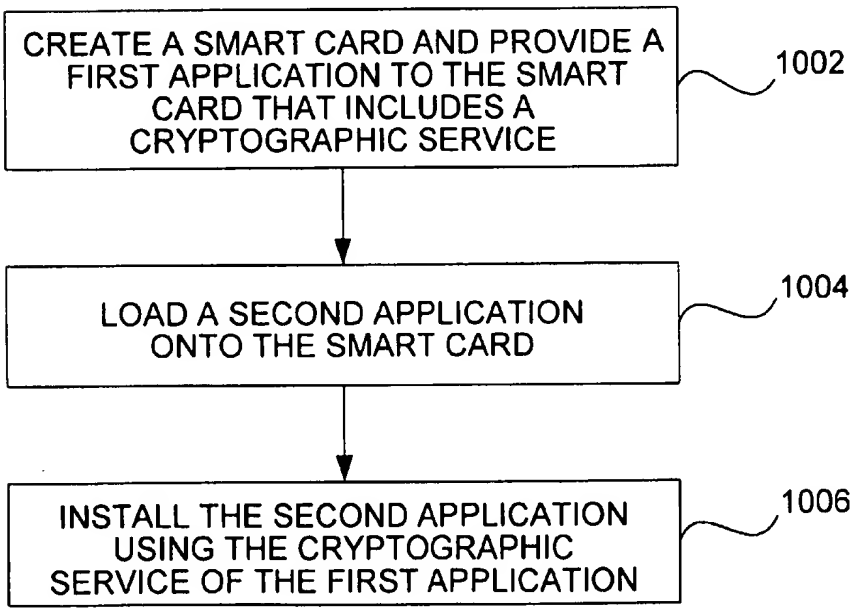

CREATE A SMART CARD AND PROVIDE A FIRST APPLICATION TO THE SMART CARD THAT INCLUDES A CRYPTOGRAPHIC SERVICE
1002
LOAD A SECOND APPLICATION ONTO THE SMART CARD
1004
INSTALL THE SECOND APPLICATION USING THE CRYPTOGRAPHIC SERVICE OF THE FIRST APPLICATION
1006

FIG. 5

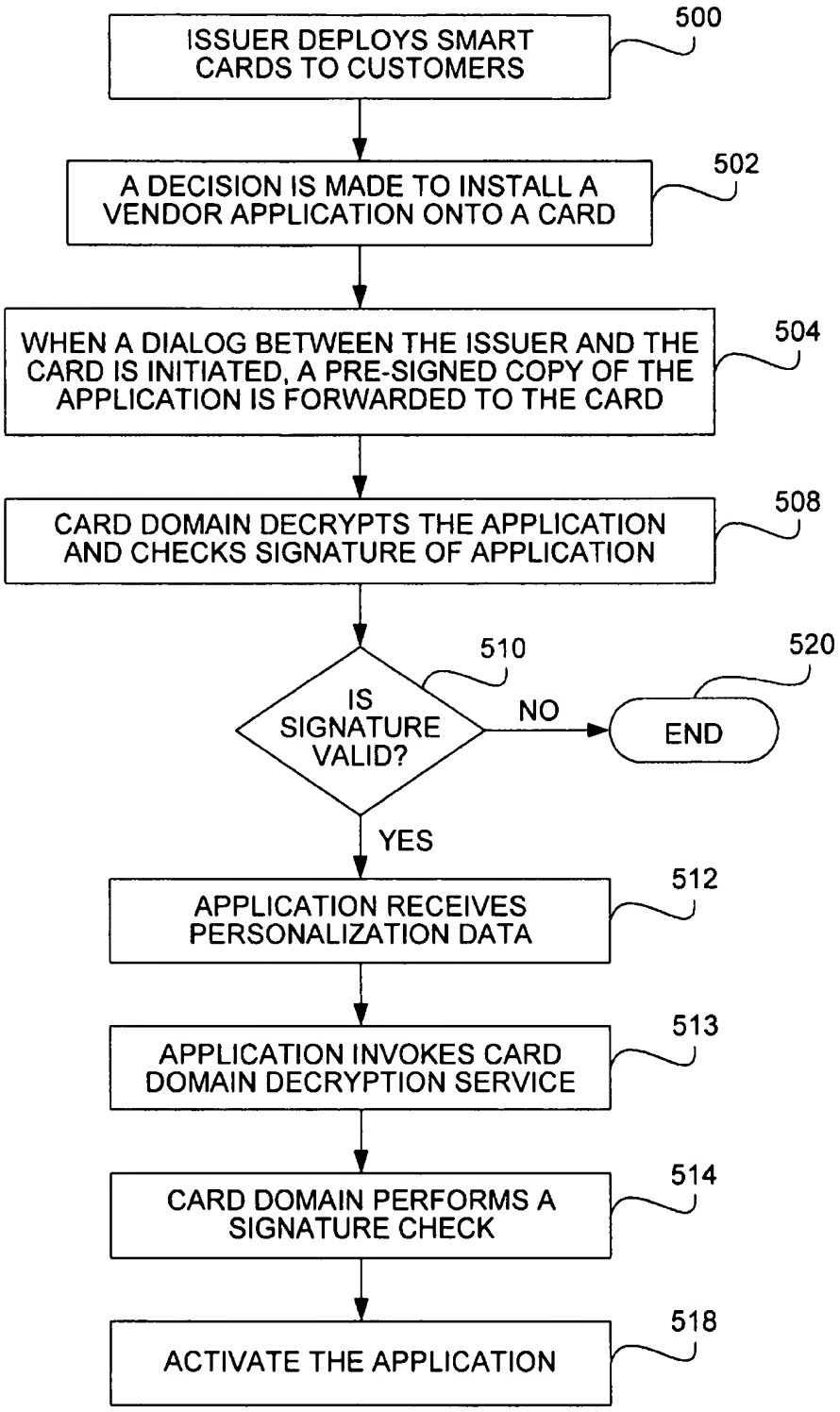

ISSUER DEPLOYS SMART CARDS TO CUSTOMERS
500
A DECISION IS MADE TO INSTALL A VENDOR APPLICATION ONTO A CARD
502
WHEN A DIALOG BETWEEN THE ISSUER AND THE CARD IS INITIATED, A PRE-SIGNED COPY OF THE APPLICATION IS FORWARDED TO THE CARD
504
CARD DOMAIN DECRYPTS THE APPLICATION AND CHECKS SIGNATURE OF APPLICATION
508
IS SIGNATURE VALID?
510
NO
END
520
YES
APPLICATION RECEIVES PERSONALIZATION DATA
512
APPLICATION INVOKES CARD DOMAIN DECRYPTION SERVICE
513
CARD DOMAIN PERFORMS A SIGNATURE CHECK
514
ACTIVATE THE APPLICATION
518

FIG. 6

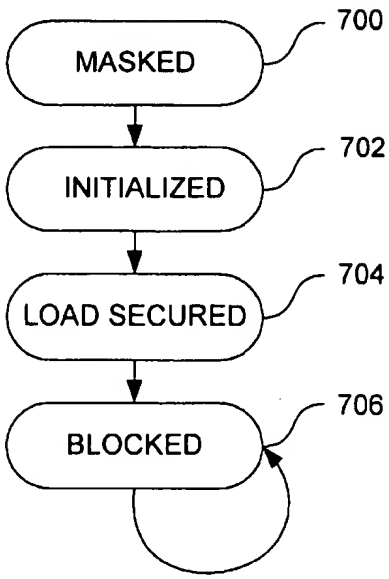

MASKED
700
INITIALIZED
702
LOAD SECURED
704
BLOCKED
706

FIG. 7A

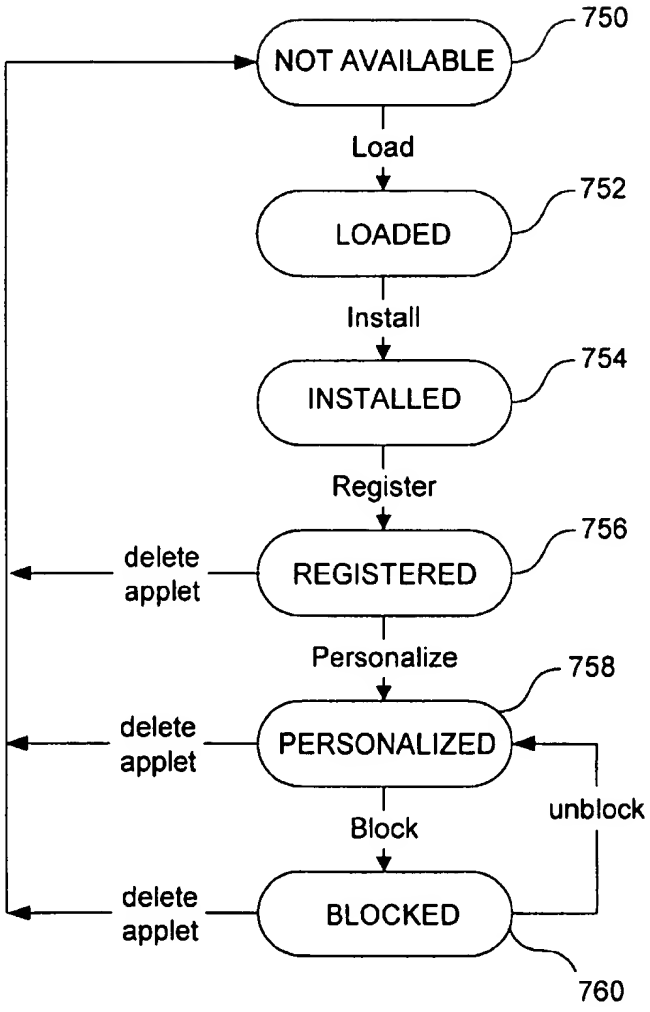

NOT AVAILABLE
750
Load
LOADED
752
Install
INSTALLED
754
Register
REGISTERED
756
delete applet
Personalize
PERSONALIZED
758
delete applet
unblock
Block
BLOCKED
delete applet
760

FIG. 7B

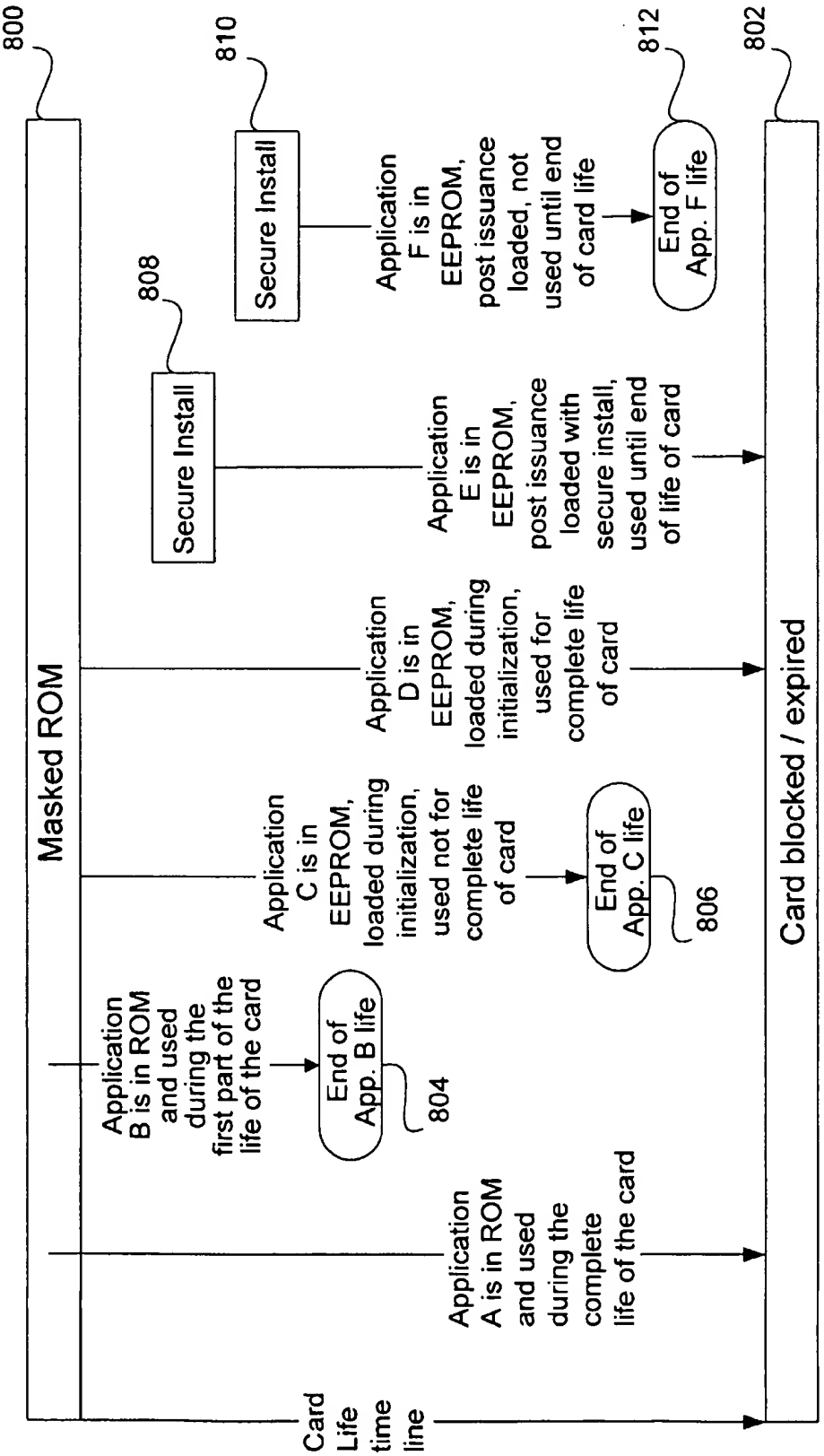

800
Masked ROM
Application A is in ROM and used during the complete life of the card
Application B is in ROM and used during the first part of the life of the card
End of App. B life
804
Application C is in EEPROM, loaded during initialization, used not for complete life of card
End of App. C life
806
Application D is in EEPROM, loaded during initialization, used for complete life of card
808
Secure Install
Application E is in EEPROM, post issuance loaded with secure install, used until end of life of card
810
Secure Install
Application F is in EEPROM, post issuance loaded, not used until end of card life
812
End of App. F life
802
Card blocked / expired
Card Life time line

FIG. 8

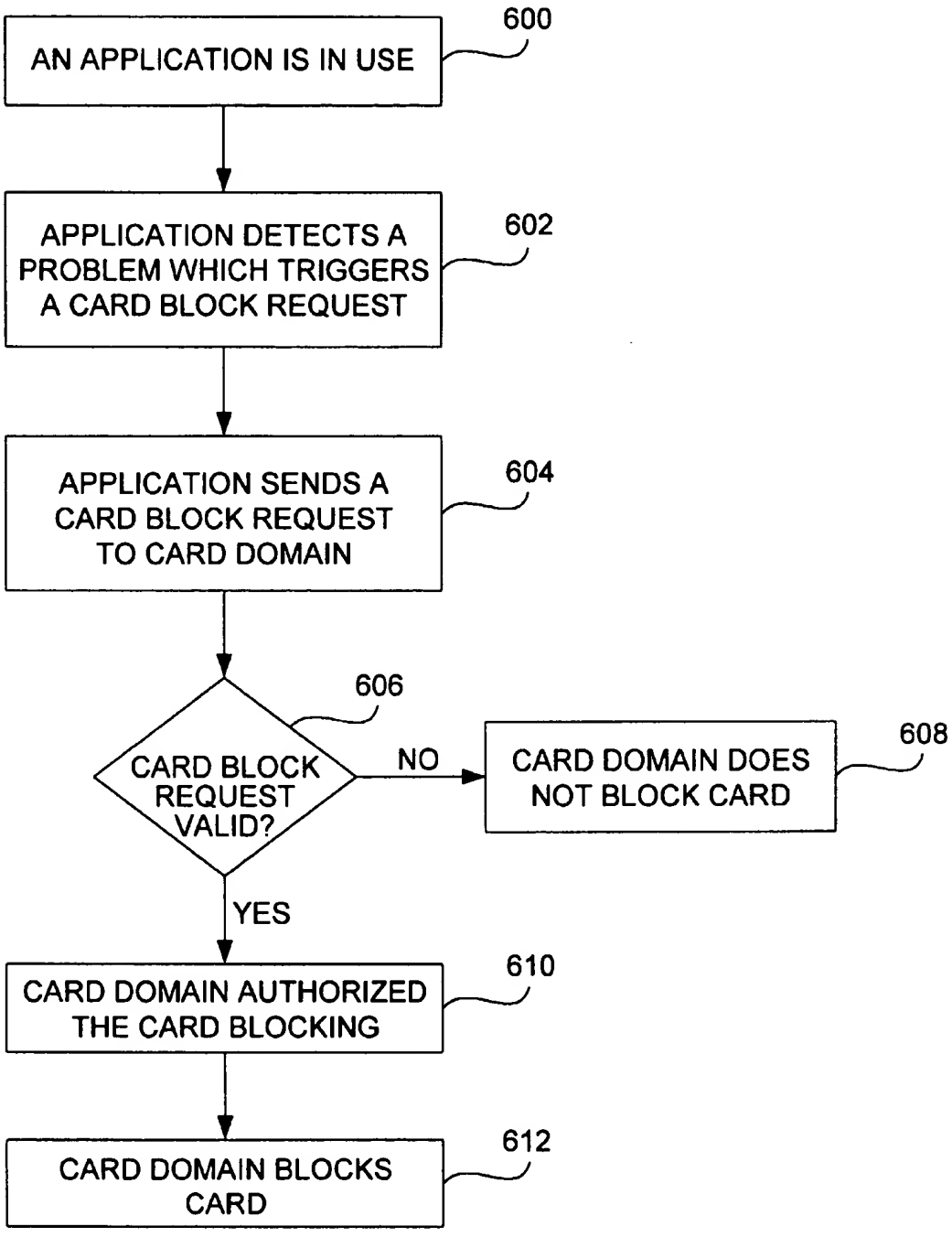

AN APPLICATION IS IN USE
600
APPLICATION DETECTS A PROBLEM WHICH TRIGGERS A CARD BLOCK REQUEST
602
APPLICATION SENDS A CARD BLOCK REQUEST TO CARD DOMAIN
604
CARD BLOCK REQUEST VALID?
606
NO
CARD DOMAIN DOES NOT BLOCK CARD
608
YES
CARD DOMAIN AUTHORIZED THE CARD BLOCKING
610
CARD DOMAIN BLOCKS CARD
612

FIG. 9

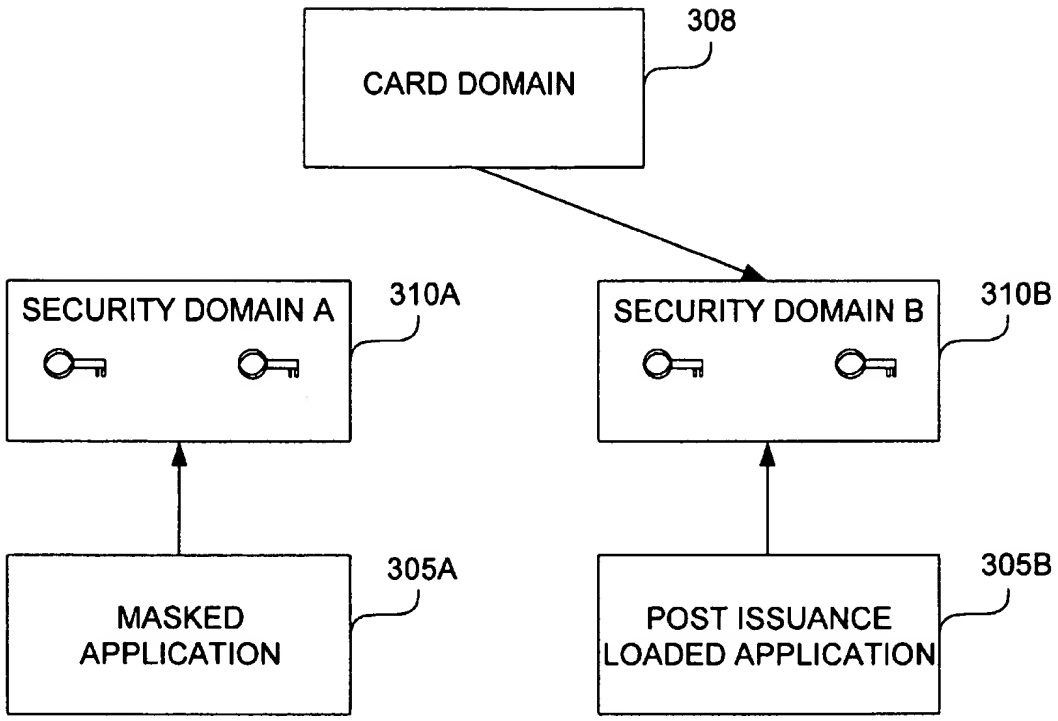

CARD DOMAIN
308
SECURITY DOMAIN A
310A
SECURITY DOMAIN B
310B
MASKED APPLICATION
305A
POST ISSUANCE LOADED APPLICATION
305B

FIG. 10

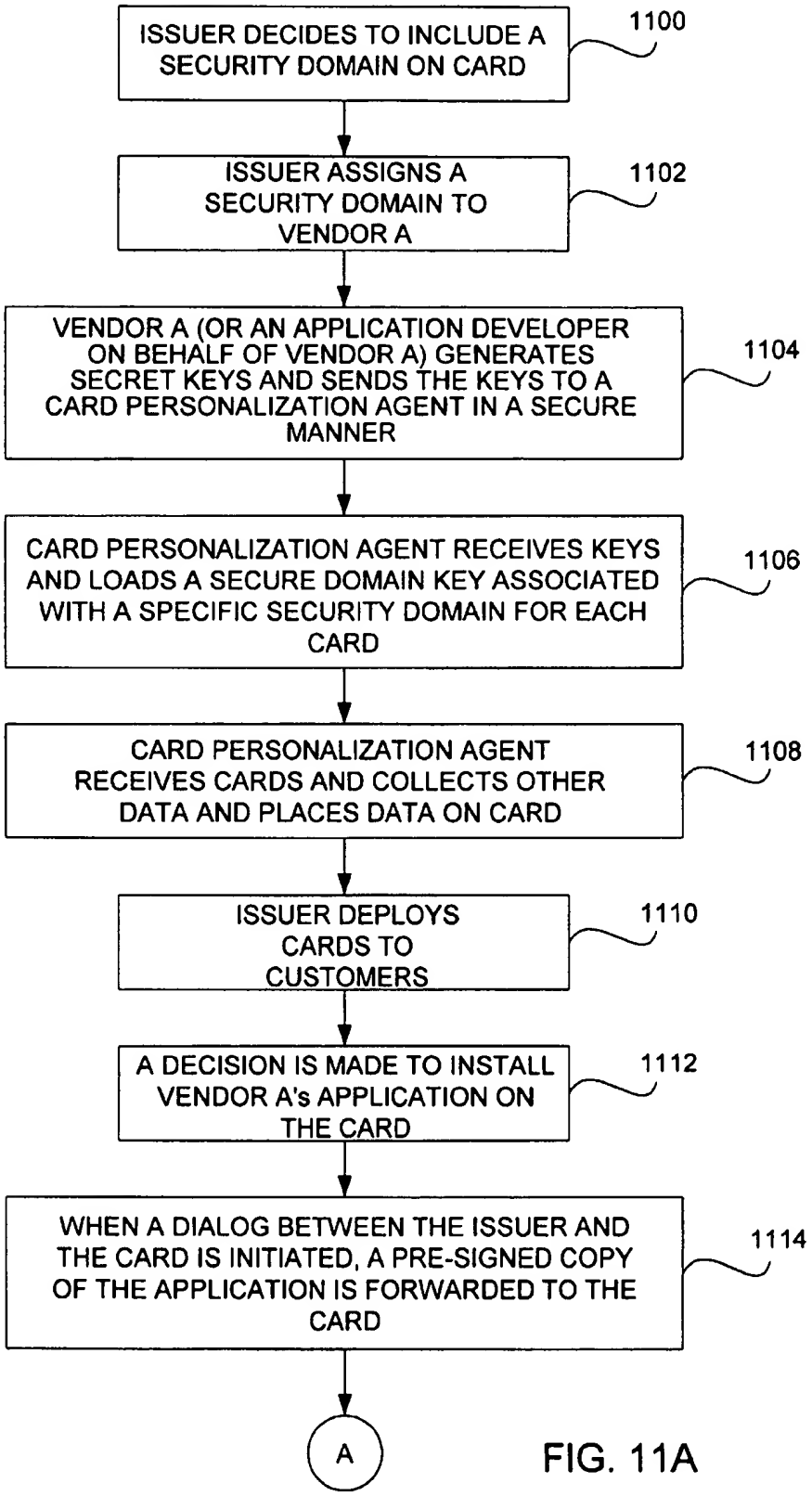

ISSUER DECIDES TO INCLUDE A SECURITY DOMAIN ON CARD
1100
ISSUER ASSIGNS A SECURITY DOMAIN TO VENDOR A
1102
VENDOR A (OR AN APPLICATION DEVELOPER ON BEHALF OF VENDOR A) GENERATES SECRET KEYS AND SENDS THE KEYS TO A CARD PERSONALIZATION AGENT IN A SECURE MANNER
1104
CARD PERSONALIZATION AGENT RECEIVES KEYS AND LOADS A SECURE DOMAIN KEY ASSOCIATED WITH A SPECIFIC SECURITY DOMAIN FOR EACH CARD
1106
CARD PERSONALIZATION AGENT RECEIVES CARDS AND COLLECTS OTHER DATA AND PLACES DATA ON CARD
1108
ISSUER DEPLOYS CARDS TO CUSTOMERS
1110
A DECISION IS MADE TO INSTALL VENDOR A's APPLICATION ON THE CARD
1112
WHEN A DIALOG BETWEEN THE ISSUER AND THE CARD IS INITIATED, A PRE-SIGNED COPY OF THE APPLICATION IS FORWARDED TO THE CARD
1114
A

FIG. 11A

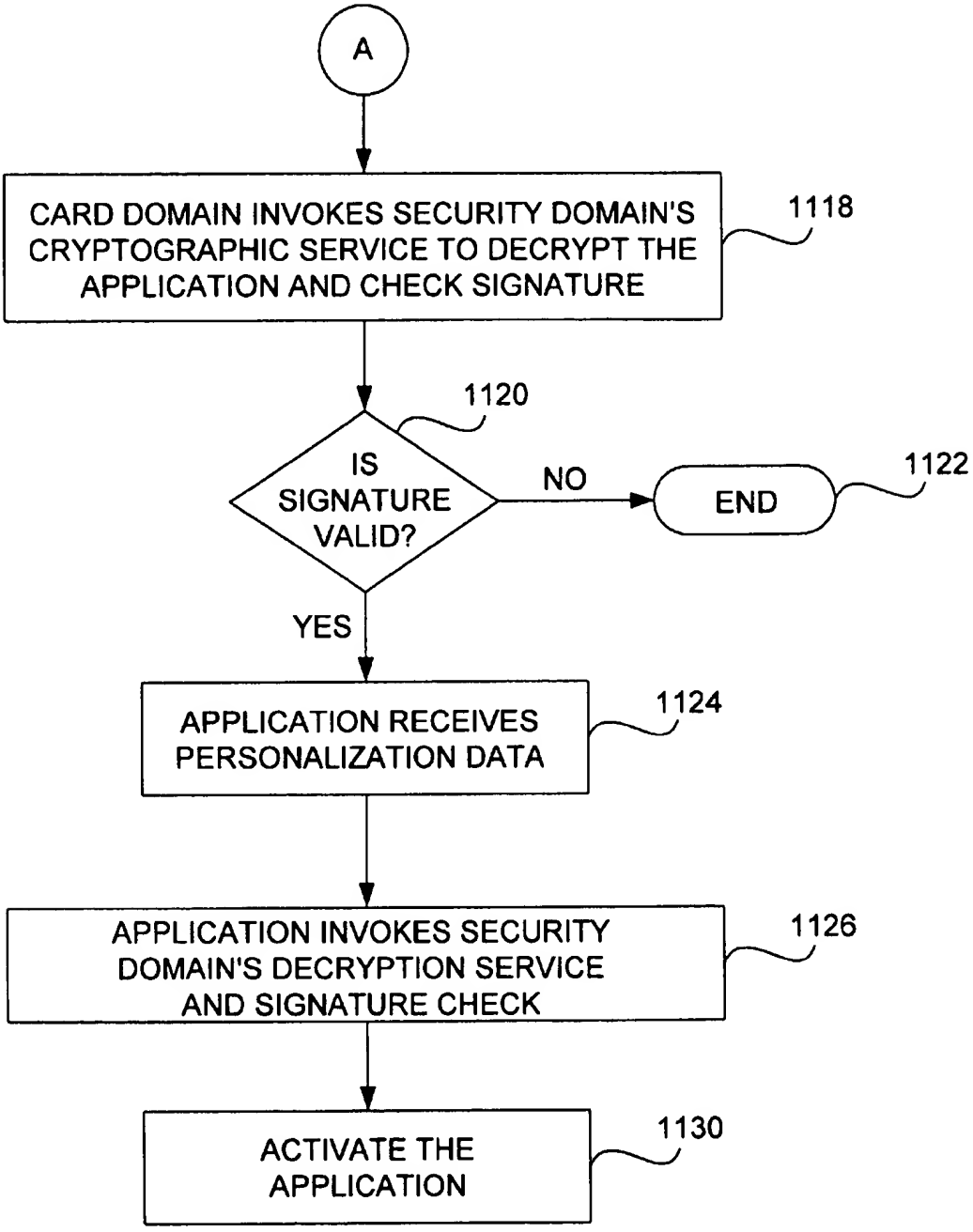

A
CARD DOMAIN INVOKES SECURITY DOMAIN'S CRYPTOGRAPHIC SERVICE TO DECRYPT THE APPLICATION AND CHECK SIGNATURE
1118
1120
IS SIGNATURE VALID?
NO
END
1122
YES
APPLICATION RECEIVES PERSONALIZATION DATA
1124
APPLICATION INVOKES SECURITY DOMAIN'S DECRYPTION SERVICE AND SIGNATURE CHECK
1126
ACTIVATE THE APPLICATION
1130

FIG. 11B

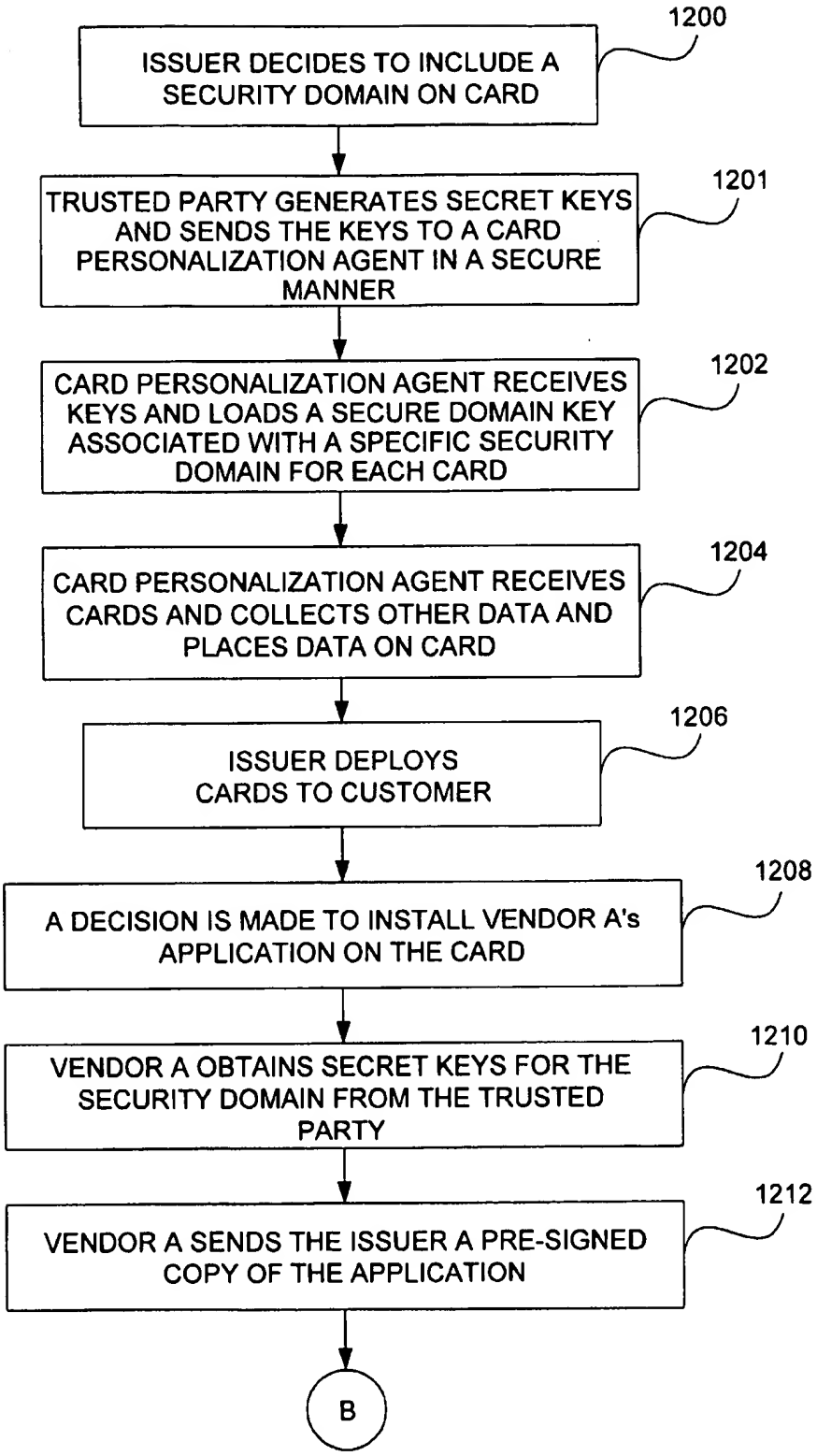

1200
ISSUER DECIDES TO INCLUDE A SECURITY DOMAIN ON CARD
1201
TRUSTED PARTY GENERATES SECRET KEYS AND SENDS THE KEYS TO A CARD PERSONALIZATION AGENT IN A SECURE MANNER
1202
CARD PERSONALIZATION AGENT RECEIVES KEYS AND LOADS A SECURE DOMAIN KEY ASSOCIATED WITH A SPECIFIC SECURITY DOMAIN FOR EACH CARD
1204
CARD PERSONALIZATION AGENT RECEIVES CARDS AND COLLECTS OTHER DATA AND PLACES DATA ON CARD
1206
ISSUER DEPLOYS CARDS TO CUSTOMER
1208
A DECISION IS MADE TO INSTALL VENDOR A's APPLICATION ON THE CARD
1210
VENDOR A OBTAINS SECRET KEYS FOR THE SECURITY DOMAIN FROM THE TRUSTED PARTY
1212
VENDOR A SENDS THE ISSUER A PRE-SIGNED COPY OF THE APPLICATION
B

FIG. 12A

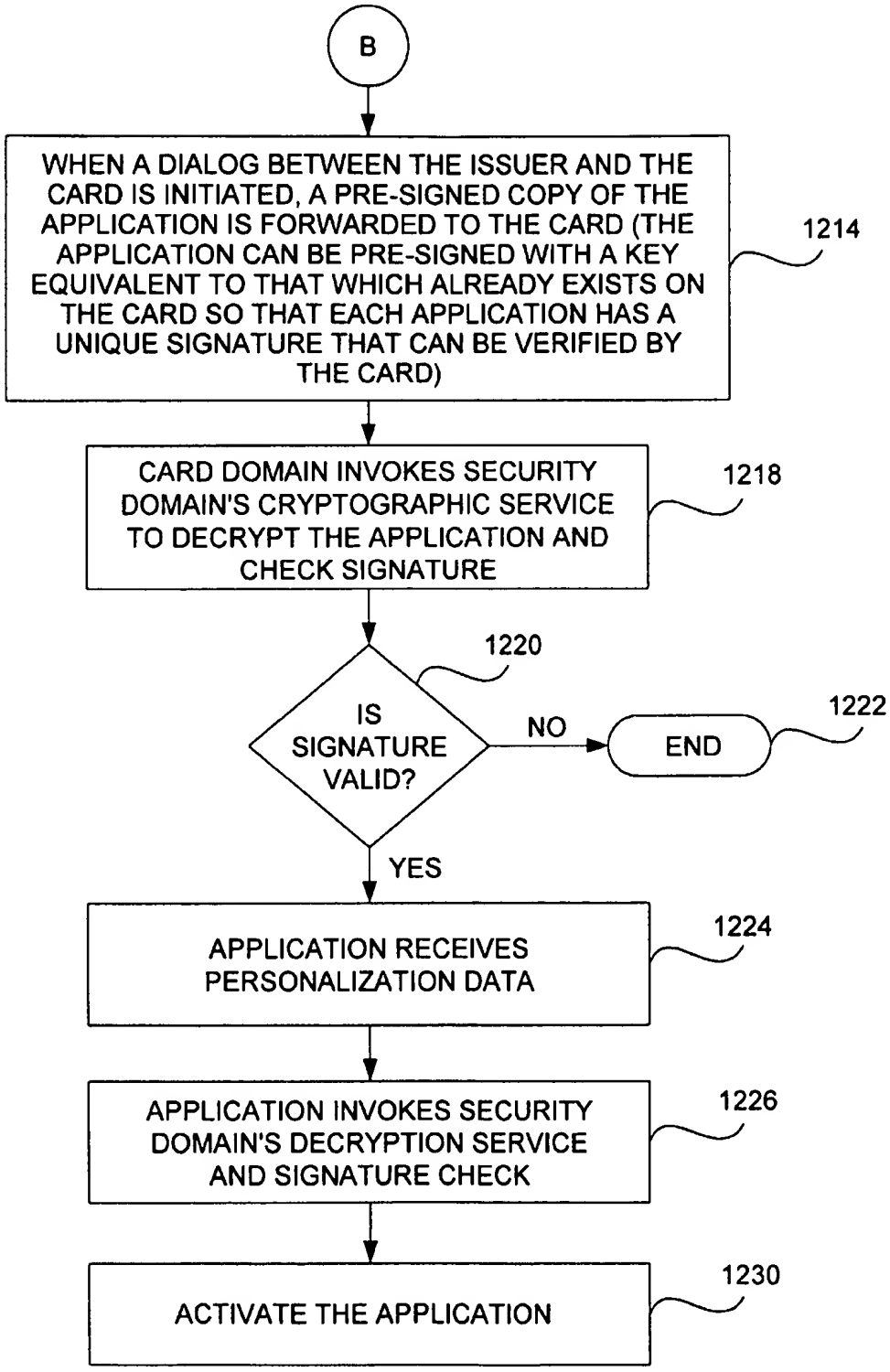

B
WHEN A DIALOG BETWEEN THE ISSUER AND THE CARD IS INITIATED, A PRE-SIGNED COPY OF THE APPLICATION IS FORWARDED TO THE CARD (THE APPLICATION CAN BE PRE-SIGNED WITH A KEY EQUIVALENT TO THAT WHICH ALREADY EXISTS ON THE CARD SO THAT EACH APPLICATION HAS A UNIQUE SIGNATURE THAT CAN BE VERIFIED BY THE CARD)
1214
CARD DOMAIN INVOKES SECURITY DOMAIN'S CRYPTOGRAPHIC SERVICE TO DECRYPT THE APPLICATION AND CHECK SIGNATURE
1218
1220
IS SIGNATURE VALID?
NO
END
1222
YES
APPLICATION RECEIVES PERSONALIZATION DATA
1224
APPLICATION INVOKES SECURITY DOMAIN'S DECRYPTION SERVICE AND SIGNATURE CHECK
1226
ACTIVATE THE APPLICATION
1230

FIG. 12B

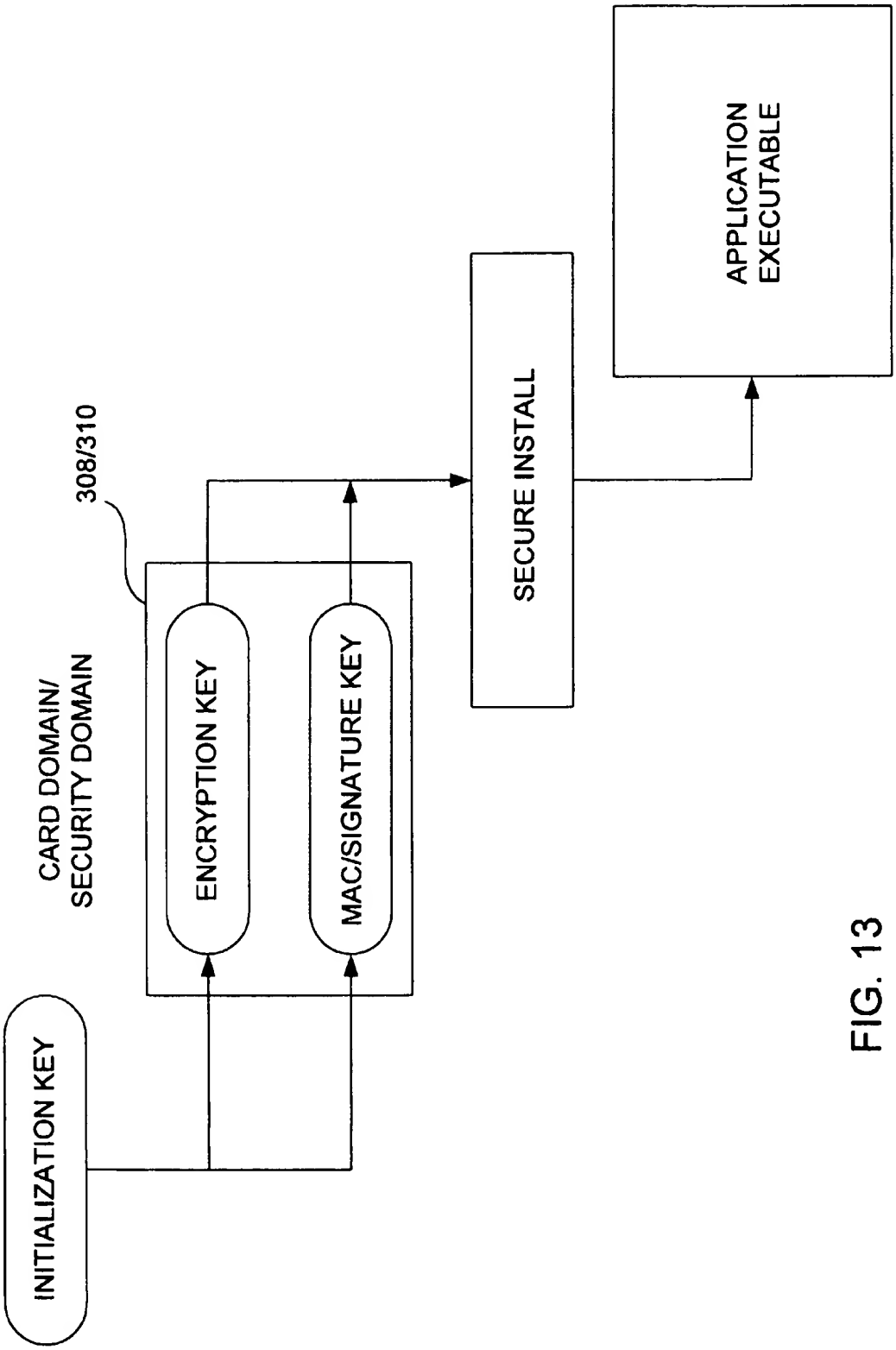

APPLICATION EXECUTABLE
SECURE INSTALL
308/310
CARD DOMAIN/ SECURITY DOMAIN
ENCRYPTION KEY
MAC/SIGNATURE KEY
INITIALIZATION KEY

FIG. 13

SYSTEM AND METHOD FOR A MULTI-APPLICATION SMART CARD WHICH CAN FACILITATE A POST-ISSUANCE DOWNLOAD OF AN APPLICATION ONTO THE SMART CARD

CROSS REFERENCE TO RELATED APPLICATION

This application claims priority to U.S. provisional application No. 60/061,763 filed Oct. 14, 1997, which is herein incorporated by reference. This application further claims priority to U.S. provisional application No. 60/041,468 filed Mar. 24, 1997, which is also herein incorporated by reference.

This application is related to U.S. application Ser. No. 09/046,993 now U.S. Pat. No. 6,005,942 (VISAP009 or VISAP010), filed on even date herewith which is also herein incorporated by reference for all purposes.

FIELD OF THE INVENTION

The present invention relates to smart cards. In particular, the present invention relates to a system and method for providing a multi-application smart card which can facilitate a post-issuance download of an application onto the smart card.

BACKGROUND OF THE INVENTION

A smart card is typically a credit card-sized plastic card that includes a semiconductor chip capable of holding data supporting multiple applications.

Physically, a smart card often resembles a traditional "credit" card having one or more semiconductor devices attached to a module embedded in the card, providing contacts to the outside world. The card can interface with a point-of-sale terminal, an ATM, or a card reader integrated into a telephone, a computer, a vending machine, or any other appliance.

A micro-controller semiconductor device embedded in a "processor" smart card allows the card to undertake a range of computational operations, protected storage, encryption and decision making. Such a micro-controller typically includes a microprocessor, memory, and other functional hardware elements. Various types of cards are described in "The Advanced Card Report: Smart Card Primer", Kenneth R. Ayer and Joseph F. Schuler, The Schuler Consultancy, 1993.

One example of a smart card implemented as a processor card is illustrated in FIG. 1. Of course, a smart card may be implemented in many ways, and need not necessarily include a microprocessor or other features. The smart card may be programmed with various types of functionality, including applications such as stored-value; credit/debit; loyalty programs, etc.

In some embodiments, smart card **5** has an embedded micro-controller **10** that includes a microprocessor **12**, random access memory (RAM) **14**, read-only memory (ROM) **16**, non-volatile memory **18**, a cryptographic module **22**, and a card reader interface **24**. Other features of the micro-controller may be present but are not shown, such as a clock, a random number generator, interrupt control, control logic, a charge pump, power connections, and interface contacts that allow the card to communicate with the outside world.

Microprocessor **12** is any suitable central processing unit for executing commands and controlling the device. RAM **14** serves as storage for calculated results and as stack memory. ROM **16** stores the operating system, fixed data, standard routines, and look up tables. Non-volatile memory **18** (such as EPROM or EEPROM) serves to store information that must not be lost when the card is disconnected from a power source but that must also be alterable to accommodate data specific to individual cards or any changes possible over the card lifetime. This information might include a card identification number, a personal identification number, authorization levels, cash balances, credit limits, etc. Cryptographic module **22** is an optional hardware module used for performing a variety of crptographic algorithms. Card reader interface **24** includes the software and hardware necessary for communication with the outside world. A wide variety of interfaces are possible. By way of example, interface **24** may provide a contact interface, a close-coupled interface, a remote-coupled interface, or a variety of other interfaces. With a contact interface, signals from the micro-controller are routed to a number of metal contacts on the outside of the card which come in physical contact with similar contacts of a card reader device.

Various mechanical and electrical characteristics of smart card **5** and aspects of its interaction with a card reading device are defined by the following specifications, all of which are herein incorporated by reference.

Visa Integrated Circuit Card Specification, (Visa International Service Association 1996).

EMV Integrated Circuit Card Specification for Payment Systems, (Visa International Service Association 1996).

EMV Integrated Circuit Card Terminal Specification for Payment Systems, (Visa International Service Association 1996).

EMV Integrated Circuit Card Application Specification for Payment Systems, (Visa International Service Association 1996).

International Standard; Identification Cards—Integrated Circuit(s) Cards with Contacts, Parts 1–6 (International Standards Organization 1987–1995).

Prior to issuance of a smart card to a card user, the smart card is initialized such that some data is placed in the card. For example, during initialization, the smart card may be loaded with at least one application, such as credit or stored cash value, a file structure initialized with default values, and some initial cryptographic keys for transport security. Once a card is initialized, it is typically personalized. During personalization, the smart card is loaded with data which uniquely identifies the card. For example, the personalization data can include a maximum value of the card, a personal identification number (PIN), the currency in which the card is valid, the expiration date of the card, and cryptographic keys for the card.

A limitation of conventional smart cards is that new applications typically can not be added to an issued smart card. Smart cards are traditionally issued with one or more applications predefined and installed during the manufacturing process of the card. As a result, with traditional smart card implementation, once a card has been issued to a card user, the smart card becomes a fixed application card. If a new application is desired, the smart card is typically discarded and a new smart card, which includes the new application, is issued.

It would be desirable to provide a smart card which would allow applications to be loaded after the card is issued. Further, it is desirable to provide a mechanism to manage the loading of an application as well as general management of the applications on the smart card. Additionally, it is desirable to allow an application provider to keep cryptographic

keys confidential from the issuer of the smart card and to securely allow application from different entities to coexist on a card.

SUMMARY OF THE INVENTION

Embodiments of the present invention teach a system and method which allows card issuers to add applications during the lifetime of the card after the card has already been issued (referred to herein as post issuance loading). The process of downloading an application after the card has been issued to the card holder will be referred to herein as a "secure install" process.

The system and method according to embodiments of the present invention allow the loading of an application and/or objects from an application server via a card acceptance device and its supporting system infrastructure delivery mechanism, onto a card, post issuance in a secure and confidential manner.

An embodiment of the present invention provides a system and method for providing confidential information to an application in a smart card. In a multi-application smart card, a privileged application, herein referred to as a security domain, is utilized as a confidential representative of an application provider. The security domain can contain cryptographic keys which can be kept confidential from the smart card issuer, thus allowing separation of cryptographic security between the issuer and the application provider. When a new application is loaded onto a smart card, the newly loaded application can utilize its associated security domain's cryptographic service. A privileged application representing the issuer, herein referred to as a card domain, can approve of commands, such as commands for initialization and personalization, by invoking the security domain's cryptographic service. In this manner, a post issuance download of an application onto the issued smart card can be accomplished.

A method according to an embodiment of the present invention for providing confidential information to an application in a smart card is presented. The method comprises the steps of providing a first application in the smart card, the first application including a cryptographic service; loading a second application onto the smart card; and installing the second application, wherein the cryptographic service of the first application is utilized to install the second application.

In another aspect of the invention, a system according to an embodiment of the present invention for providing confidential information to an application in a smart card is presented. The system comprises a first application associated with the issued smart card, wherein the first application includes cryptographic service; and a second application associated with the issued smart card, the second application being in communication with the first application, wherein the cryptographic service included in the first application is utilized for at least one function related to the second application. In yet another aspect of the invention, a method according to an embodiment of the present invention for providing an application to a smart card is presented. The method comprising the steps of issuing a smart card; loading a first application onto the issued smart card; and initializing the first application.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a smart card system suitable for implementing the present invention.

FIG. **2** is an example of a block diagram of software layers which can be utilized in a smart card.

FIGS. **3A–3B** are block diagrams of examples of software layers according to embodiments of the present invention.

FIG. **4** is a flow diagram of an example of a method according to an embodiment of the present invention for installing an application onto an issued smart card utilizing a card domain.

FIG. **5** is a flow diagram of a method according to an embodiment of the present invention for providing confidential information to an application in a smart card using security domains.

FIG. **6** is a flow diagram of an example of a method according to an embodiment of the present invention for installing an application onto an issued smart card utilizing a card domain.

FIG. **7A** is a flow diagram illustrating a sequence of card life states.

FIG. **7B** is a flow diagram illustrating a sequence of card life states.

FIG. **8** is an illustration of an example of a card life cycle.

FIG. **9** is a flow diagram of an example of a method according to an embodiment of the present invention for blocking a card utilizing a card domain.

FIG. **10** is a block diagram illustrating interactions between a card domain and a security domain on a smart card according to an embodiment of the present invention.

FIGS. **11A** and **11B** are flow diagrams of an example of a method according to an embodiment of the present invention for loading an application by using a security domain after the smart card has issued.

FIGS. **12A–12B** are flow diagrams of an example of a method according to an alternate embodiment of the present invention for loading an application using a security domain after the smart card has issued.

FIG. **13** is a block diagram illustrating an example of key management and key dependencies for post issuance download of applications onto the smart card.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The following description is presented to enable one of ordinary skill in the art to make and to use the invention and is provided in the context of a patent application and its requirements. Various modifications to the preferred embodiments will be readily apparent to those skilled in the art and the generic principles herein may be applied to other embodiments. Thus, the present invention is not intended to be limited to the embodiment shown but is to be accorded the widest scope consistent with the principles and features described herein.

FIG. **2** is a block diagram of an example of software layers which can be utilized in a smart card. The smartcard shown in FIG. **2** includes an operating system **200**, a card application programming interface (API) **204**, and applications **206A–206B**. Operating system **200** can include functionality to control the cards, memory management, input/output (I/O), and cryptographic features. Card API **204** utilizes the instructions from operating system **200** and writes these instructions into blocks which can be reused for common routines in multiple applications. Applications **206A** and **206B** can run on the smart card via instructions from API **204**. These applications can include any application which can run on a smart card, such as stored value, credit, debit, transit, and loyalty.

One embodiment of the present invention is based upon the Java Card standard. In this case applications are referred

to as 'Applets' and they are written to link to a Java Card API which is the application programming interface present on smart cards built to the Java Card standard.

Although the conventional software system shown in FIG. **2** allows for multiple applications, it does not solve the problem of how to load, securely, an application after issuance of the smart card to a user. If an application is to be loaded post issuance, a mechanism is needed to manage the loading of an application as well as general management of the applications on the smart card. Additionally, an application provider may wish to keep cryptographic keys confidential from the issuer of the smart card. Accordingly, a mechanism is needed to provide for the separation of confidential information between an application provider and an issuer of a smart card. Embodiments of the present invention address such a need.

FIGS. **3**A–**3**B are block diagrams showing software components of a smart card according to embodiments of the present invention. The arrows indicate dependencies between components. FIG. **3**A shows an embodiment of a smart card utilizing a card domain, while FIG. **3**B shows an embodiment of a smart card utilizing a security domain, as well as a card domain.

The example shown in FIG. **3**A includes an operating system **300**, a card API **304**, applications **305**A–**305**C, a card domain **308**, and open platform (OP) API **306**. The system shown in FIG. **3** allows for a secure and managed post issuance download of an application onto a smart card.

Open platform API **306** classifies instructions into card domain **308** and security domains **310**A–**310**B (shown in FIG. **3**B). Accordingly, OP API **306** facilitates the formation of instructions into sets which can be identified as being included as part of card domain **308** and security domains **310**A–**310**B.

Applications **305**A–**305**C can include any application which can be supported by a smart card. Examples of these applications include credit, debit, stored value, transit, and loyalty. Applications **305**A–**305**C are shown to include command interfaces, such as APDU interfaces **354**A–**354**C which facilitate communication with the external environment.

Applications **305**A and **305**B can run on the smart card via instructions from card API **304**. Card API **304** is implemented using the instructions from the card operating system and writes these instructions into blocks which can be reused for common routines for multiple applications. Those skilled in the art will recognize that a translation layer or interpreter may reside between API **304** and operating system **300**. An interpreter interprets the diverse hardware chip instructions from vendor specific operating system **300** into a form which can be readily utilized by card API **304**.

Card domain **308** can be a "privileged" application which represents the interests of the smart card issuer. As a "privileged" application, card domain **308** may be configured to perform multiple functions to manage various aspects of the smart card. For instance, card domain **308** can perform functions such as installing an application on the smart card, installing security domain s **310**A–**310**B (shown on FIG. **3**B), personalization and reading of card global data, managing card life cycle states (including card blocking), performing auditing of a blocked card, maintaining a mapping of card applications **305**A–**305**C to security domains **310**A–**310**B, and performing security domain functions for applications **305**A–**305**C which are not associated with a security domain **310**.

Card domain **308** is shown to include an API interface **350** and a command interface, such as Application Protocol Data Unit (APDU) interface **352**. APDU interface **352** facilitates interfacing with the external environment. In compliance with, e.g., International Standards Organization (ISO) Standard 7816-4, entitled "Identification Cards—Integrated circuit(s) cards with contacts—Part 4, Inter-industry commands for interchange," which is herein incorporated by reference.

For example, APDU interface **352** can be used during post issuance installation of an application or during loading of card global data. An application load and install option is performed via a set of appropriate APDU commands received by card domain **308**. API interface **350** facilitates interfacing with the internal smart card environment. For example, API interface **350** can by used if card domain **308** is being utilized as a default in place of a security domain **310**, or if an application requires information such as card global data, key derivation data, or information regarding card life cycle.

Memory allocations have been performed by the time an application is in an install state. An application is also personalized after loading and installing. A personalized application includes card holder specific data and other required data which allows the application to run. In addition to managing the installation and personalization of the application, card domain **308** can also manage global card information. Global card information includes information that several applications may need to perform their functions, such as card holder name and card unique data utilized in cryptographic key derivations. Card domain **308** can be a repository for the global card information to avoid storing the same data multiple times.

Card domain **308** can also manage card life cycle states including card blocking. The smart card will typically move through several states during its life cycle. Card domain **308** keeps track of what state the card is in during its life cycle. Card domain **308** may also manage a block request to block virtually all functions of the card. Further details of card domain **308** management of a block request will be discussed in conjunction with FIG. **6**. Card domain **308** may also keep track of the state of an application during an application's life cycle. This kind of information regarding an application can be utilized during an auditing of a card. Auditing can be performed at any time during a card's lifetime. For instance, auditing may be performed after a card has been blocked or prior to installing a new application to validate the card contents. Although virtually all card functions are no longer functioning when a card is blocked, an issuer may be able to query card domain **308** for information regarding a state of an application or the life cycle state of the card. In this manner, the issuer of a card may still access a profile of the blocked card and its applications.

FIG. **3**B shows an embodiment of the present invention utilizing a security domain **310**, as well as card domain **308**. The example shown in FIG. **3**B includes a operating system **300'**, a card API **304'**, applications **305**A–**305**C', security domains **310**A–**310**B', a card domain **308'**, and open platform (OP) API **306'**. The system shown in FIG. **3**B also allows for a secure and managed post issuance download of an application onto a smart card.

Card domain **308'** can work in conjunction with a security domain **310**. Security domain **310** is a logical construct that can be implemented as an application to provide security related functions to card domain **308'** and to applications associated with security domain **310**. Security domains **310**A–**310**B can assist in secure post issuance loading of an

application onto the smart card. Security domains **310A–310B** provide for a mechanism which keeps the application provider's confidential information, such as cryptographic keys, from being disclosed to the issuer of the smart card.

There may be multiple security domains **310** on a smart card, each represented by a unique cryptographic relationship. A security domain **310** is responsible for the management and sharing of cryptographic keys and the associated cryptographic methods which make up the security domain's cryptographic relationship. An application which is loaded to the smart card post issuance can be associated with a security domain, preferably with only one security domain. However, multiple applications may be associated with the same security domain **310**. Applications installed on a smart card during the pre-issuance phase may optionally be associated with a security domain **310** on the smart card for purposes of loading confidential personalization data to those applications using security domain **310** keys.

The software for security domain **310** may be installed by the card manufacturer at the time of card manufacturing (e.g., when the ROM is masked), or may be added during initialization or personalization stages. Security domains **310** can be implemented as selectable applications which are isolated from one another and the rest of the system. If security domain **310** is implemented in a Java card as an application, standard Java card security can be relied upon to ensure isolation of security domain **310**. In addition, or alternatively, other security mechanisms such as hardware security which can be utilized through OP API **306** implementation. OP API **306** may utilize special security features to enforce isolation of security domain **310**. An example of such a security feature is the utilization of chip hardware security routines which may be employed by OP API **306**.

Each security domain **310A–310B** provides a command interface, such as an Application Protocol Data Unit (APDU) interface **320A–320B**, for communication off card and an on card API interface **322A–322B**.

The APDU interface **320A–320B** consists of personalization commands and is intended to allow the initial loading of security domain keys and to support key rotation if desired during the life of the security domain. API interfaces **322A–322B** may include a signature verification method and decryption method which are shared with card domain **308'** for post issuance loading of applications. Additionally, applications may utilize API interfaces **322A–322B** for decrypting application confidential data. Note that card domain **308'** may always function as a security domain and does so as the default.

Security domain **310** manages signing and decrypting keys and provides cryptographic services using those keys. Security domain **310** processes APDU's for numerous functions. These functions can include key management functions e.g., functions to load or update keys. During Secure Installation of an application, security domain **310** can provide services to card domain **308'** to decrypt an application install file and check the signature of an application file. For an application associated with a security domain **310**, that application's security domain **310** provides decrypt and signature functions, such as MACing on an update key APDU command during the personalization phase of a newly installed application. Thereafter, the application can use the updated key to decrypt and check signatures on subsequent key updates.

The smart card issuer may decide whether security domain **310** utilizes a static key or a session key for transactions. A static key is a cryptographic key which exists prior to processing APDUs and which exist during and after the processing of APDUs. A session key is a cryptographic key which can be generated for a particular transaction and is typically no longer used for APDU processing after the transaction. If a session key is utilized, security domain **310** preferably derives its own session key for processing APDUs.

FIG. **4** is a flow diagram of a method accordingly to an embodiment of the present invention for providing an application onto a smart card. The example illustrated in FIG. **4** also applies to installing a security domain **310** onto a smart card. Note that all of the flow diagrams in this application are merely examples. Accordingly, the illustrated steps of this and any other flow diagram herein, can occur in various orders and in varying manners in order to accomplish virtually the same goal.

A smart card is issued (step **400**), and an application is forwarded to the issued smart card (step **402**). The forwarding of the application can occur through any electronic media which can interface with a smart card and connect to an appropriate network. For example, devices such as an automatic teller machine (ATM), a display phone, or a home computer, can be used to forward an application to the issued smart card. The forwarded application is then loaded onto the smart card, wherein the loading of the application is managed by card domain **308** (step **404**).

FIG. **5** is another flow diagram of a method according to an embodiment of the present invention for providing an application onto an issued smart card. A smart card is created and provided with a first application, the first application including a cryptographic service (step **1002**). A second application is loaded onto the smart card (step **1004**). Thereafter, the second application is installed, wherein the cryptographic service of the first application is utilized to install the second application (step **1006**).

FIG. **6** is another flow diagram of an example of a method according to an embodiment of the present invention for providing an application onto an issued smart card. This method for providing an application also applies to providing a security domain **310** onto the smart card. In the example shown in FIG. **6**, a card issuer deploys smart cards to customers (step **500**). A decision is made to install vendor A's application onto the issued smart card (step **502**). When a dialogue between the issuer and the smart card is initiated, a pre-signed copy of the application is forwarded to the smart card (step **504**). As previously stated, the dialogue between the issuer and the smart card can occur via any electronic device which can interface with a smart card and connect to an appropriate network. The application can be pre-signed with a key equivalent to that which already exists on the card so that each application has a unique signature that can be verified by the card.

Card domain **308** can then take the steps to load the application. Card domain **308** decrypts the forwarded application and checks the signature of the application (step **508**). Card domain **308** can decrypt the application with the issuer's secret key. An appropriate cryptography method, such as Data Encryption Standard (DES) or 3DES, can be utilized to decrypt at least a portion of the application. Those skilled in the art will recognize that a number of cryptographic techniques may be used to implement embodiments of the present invention. For the purpose of illustration, symmetric key techniques are addressed herein, although asymmetric techniques are also contemplated. A good general cryptography reference is Schneier, Applied

Cryptography, 2d Ed. (John Wiley, 1996), the contents of which are incorporated herein by reference.

It is then determined whether the signature on the application is valid (step **510**). If the signature associated with the application is not valid, then the application is not loaded onto the card and the process ends (step **520**). If, however, the signature associated with the application is valid the application is then installed and available for personalization. During personalization the application receives personalization data (step **512**). Personalization data includes data which is unique to the smart card user. For instance, in a airline loyalty application, personalization data can include the smart card user's seating preference, meal preference, and eligibility for various possible perks. This personalization data can also be signed and encrypted.

The application then invokes card domain's **308** decryption service (step **513**). Card domain **308** can then performs a signature check (step **514**). Methods of decrypting personalization data and performing signature checks are well known in the art. Finally, the application can then be activated (step **518**).

A new application which as been downloaded onto a smart card post-issuance can be stored in a variety of ways. One example is to store the application into a file. Another example is to maintain a pointer to the application object.

FIG. **7A** is a flow diagram illustrating an example of a sequence of card life states. The sequence is preferably considered irreversible. The first card life state is when the smart card is Masked (**700**). During the Masked state (**700**), the smart card obtains its operating system, card identification, and preferably at least one application. The Masked state (**700**) is achieved as soon as all of the necessary components for card initialization are made available. An example of when necessary components are made available is when card domain **308** and OP API **306** are enabled, as well as the Java card environment being enabled, such as Java card virtual machine **302** and Java card API **304** (both of FIG. **3**).

After the Masked state, the next state is the Initialized (step **702**) state. The Initialized state is achieved once all card activity requiring an initialization key is complete. As part of card initialization, if not already available, the card domain **308** application must be installed and registered. In addition, one or more security domains may also be installed and registered. These installed domains must then be selected and personalized. An initialization key is a secret key which is typically used by a smart card manufacturer during loading of data onto the smart card prior to issuance.

The next state is Load Secured (step **704**). The Load Secured state is achieved after a secure install (post-issuance download) mechanism for loading of applications through the remainder of the card lifetime has been established.

The final card life state is when the card is either expired or blocked (step **706**). The blocked state is achieved as soon as an authorized smart card application has received a command to block the card.

The card life cycle is preferably an irreversible sequence of states with increasing security. Initialization and all subsequent card life cycle states and their transition are preferably under the control of card domain **308**. Card domain **308** executes and responds to commands that result in a transition in a card life cycle from one state to the next. These commands are preferably Application Protocol Data Unit (APDU) commands. Card domain **308** is also responsible for the installation of applications on the card, but preferably has no control over the applications' life cycle states. Each application is preferably responsible for its own application life cycle state management but it preferably allows card domain **308** to have access to its life cycle states for auditing purposes.

The Card Life cycle is designed in such a way to increase the level of security enforced by the card at each successive state. As stated above, the cycle is also established as a process which can only ratchet forward to ensure that once the card begins a life cycle state with associated security policies, the only option is to cycle forward to the next state in the life cycle with a higher level of security. The Card Domain as the system security manager of the card maintains the current life cycle state, enforces the associated security policies, and controls the state transitions in the Card life cycle.

FIG. **7B** is a flow diagram illustrating an example of a sequence of an application life cycle. The application is initially unavailable (step **750**). The next state is a loaded state (step **752**). The application reaches the loaded state once the application has been loaded onto the smart card. The application is then installed (step **754**), and registered (step **756**). Once the application is registered, it can be deleted at any time thereafter. The next state is the personalized state, wherein personalized information is included in the application (step **758**). Finally, the application may expire or be blocked (step **760**).

FIG. **8** is an illustration of an example of multi-application card life time line. This time line starts with a Masked ROM stage **800** and ends with a card blocked/expired stage **802**. At Masked ROM stage **800**, applications A, B, C and D are shown to be installed. This example shows applications A and B being installed at a masking stage of the card, applications C and D being installed at initialization stage, and applications D and F being installed post issuance.

In this example, application A can be installed in ROM and used during the complete life of the card from Masked ROM stage **800** to card blocked/expired stage **802**. Application B is also in ROM and utilized during a first portion of the life of the smart card. The life of application B is ended at stage **804**A. Application C is located in non-volatile memory, such as EEPROM, which is loaded during initialization. Application C is shown to expire at stage **804**B. Application D is also located in EEPROM and is used for the complete life of the card until card blocked/expired stage **802**. Application E is installed at stage **806**A, sometime after issuance of the smart card. Application E is located in EEPROM and used until the end of the card life at card blocked/expired stage **802**. Application F is also installed post issuance at stage **806**B, and expires sometime before the end of the card life at stage **804**C.

FIG. **9** is a flow diagram of a method according to an embodiment of the present invention for blocking a card. A card be can be blocked if a breach of security is detected by an application. According to an embodiment of the present invention, a smart card can be blocked while an application is in use. A blocked card will no longer operate so that a suspect user cannot utilize any of the applications on the smart card. Blocking is merely one example of the many functions card domain **308** can perform in managing the other applications on the smart card. Examples of other functions include installing an application on the smart card, installing security domains **310**A–**310**B, personalization and reading of card global data, managing card life cycle states including card blocking, performing auditing of a block card, maintaining a mapping of card applications to security domains, and performing security domain functions for applications which are not associated with a security domain.

In the example shown in FIG. **9**, an application is currently in use (step **600**). The application detects a problem which triggers a card block request from the application (step **602**). The application then sends a card block request to card domain **308** (step **604**). Card domain **308** determines whether the card block request is valid (step **606**). A card block request can be valid if the request originates from a predetermined application. If the card block request is not valid, the card domain **308** does not block the smart card (step **608**). However, if the card block request is valid, then card domain **808** authorizes the card blocking (step **610**), and card domain **308** blocks the smart card (step **612**) such that the smart card will reject any attempted transactions for any of the applications on the card.

FIG. **10** is a block diagram illustrating the use of security domain **310** by the card domain **308**. The method and system according to an embodiment of the present invention allows for multiple application providers to be represented on a smart card in a secure and confidential manner. This security and confidentiality can be achieved through the use of security domain **310A–310B** shown in FIG. **3**.

FIG. **10** illustrates an example of a smart card which contains two security domains **310A–310B**. In this example, it is assumed that a masked application **305A** from the smart card is associated with a security domain, such as security domain **310A**, and an additional application **305B** will be added post issuance and be associated with a second security domain, such as security domain **310B**. The arrows indicate key relationships between the various smart card entities. Masked application **305A** uses key services from security domain **310A** for decrypting confidential data and optionally for full personalization. Card domain **308** uses key services from security domain **310B** for decrypting and checking the signature of an application loaded post issuance, such as post issuance loaded application **305B**. Post issuance loaded application **305B** uses key services from security domain **310B** for decrypting confidential data and optionally for full personalization.

FIGS. **11A** and **11B** are further flow diagrams of an example for a method according to an embodiment of the present invention for providing an application onto an issued smart card. The card issuer decides to include a security domain **310** onto a smart card (step **1100**). The issuer assigns security domain **310** to vendor A (step **1102**). Vendor A, or an application developer on behalf of vendor A, generates cryptographic keys such as those used in symmetric or asymmetric cryptography operations (step **1104**). Examples of these cryptography operations include encryption, decryption, MACing, Hashing, and digital signatures. Examples of cryptographic methods which utilize such keys and are suitable for implementation for the embodiment of the method and system of the present invention include Data Encryption Standard (DES) and 3DES. The card personalization agent receives the keys and loads security domain keys associated with a specific security domain **310** for each smart card (**1106**). The card personalization agent receives smart cards and collects other data, such as application and card holder specific data, and places data on the smart card (step **1108**).

The card issuer then deploys the smart card to customers (step **1110**). A decision is then made to install vendor A's application on the smart card (step **1112**). When a dialogue between the smart card issuer and the smart card is initiated, a signed copy of the application is forwarded to the smart card (step **1114**). The application can be signed with a key equivalent to that which already exists on the smart card so that each application has a unique signature that can be verified by the smart card.

The smart card's card domain **308** then takes steps to load the application. Card domain **308** invokes an associated security domain's cryptographic service to decrypt the application and check the signature (step **1118**). It is then determined if the signature is valid (step **1120**). If the signature is not valid, the process ends (step **1122**). If, however, the signature is found to be valid, then the application receives personalization data which can be signed and optionally encrypted (step **1124**). The loaded application then invokes its associated security domain's decryption service and signature check (step **1126**). Secret keys required to run or operate the application on the smart card are used to activate the application by authentication (step **1130**).

FIGS. **12A** and **12B** are flow diagrams of a method according to another embodiment of the present invention for providing confidential information to an application using a security domain **310**. The issuer decides to include a security domain **310** on a smart card (step **1200**). A trusted party generates secret cryptographic keys and sends the keys to a card personalization agent in a secure manner (step **1201**). A trusted party is typically a third party who performs the function of certifying the source of information, such as a signature. A card personalization agent (which may be the same as the trusted party) receives the key and loads a unique secure domain key associated with a specific security domain **310** for each smart card (step **1202**).

The card personalization agent receives the smart card and collects other data, such as application and card holder specific data, and places the data on the smart card (step **1204**). The issuer then deploys the smart card to its customers (step **1206**). A decision is made to install vendor A's application on the issued smart card (step **1208**). Vendor A obtains secret keys for security domain **310** from the trusted party (step **1210**). Vendor A then sends the smart card issuer a signed copy of Vendor A's application (step **1212**).

When a dialogue between the smart card issuer and the smart card is initiated, a signed copy of the application is forwarded to the smart card (step **1214**). The application can be signed with a key equivalent to that which already exists on the smart card so that each application has a unique signature that can be verified by the smart card. Card domain **308** invokes security domain's cryptographic service to decrypt the associated application and check its signature (step **1218**). It is then determined whether the signature is valid (step **1220**). If the signature is not valid, then the process ends (step **1222**).

If, however, the signature is valid, then the application receives personalization data, which can be signed and optionally encrypted (step **1224**). The loaded application then invokes security domain's decryption service and signature check (step **1226**). The cryptographic secret data required to run or operate the application on the card are used to activate the application (step **1230**).

FIG. **13** is a block diagram illustrating the use of cryptographic keys for post issuance loading of an application onto a smart card. Applications that are not masked and not loaded during card initialization stage or personalization stage need their executables downloaded using a secure installation method, such as the post issuance download described in previous Figures. The applications can be loaded using the card domain cryptographic keys. The applications are then decrypted and can have their signature verified using the key services of the corresponding security domain **310**. Therefore, the desired security domain(s) **310** preferably have encryption and signature keys installed prior to the post issuance download of the corresponding application.

In the example shown in FIG. **13**, only one security domain **310** is shown since security domains **310** for other applications are not relevant to illustrate the downloading of a single application. Note that the result of the secure installation is initially a loaded application, which must then be installed, registered and personalized. After loading, the application is installed, preferably by issuing an install APDU command to card domain **308**. An application can be installed when its install method has executed successfully. Memory allocations have been performed by the time an application is in an install state. A loaded application should also be registered. When an application is registered, it is selectable and it is ready to process and respond to APDU commands. Installation and registration may be performed simultaneously by the same APDU command. An application is also personalized after loading. A personalized application includes card holder specific data and other required data which allows the application to run.

In the example shown in FIG. **13**, the cryptographic key and MAC/Signature key are shown to be included in the functions of card domain **308**/security domain **310**. If a security domain is associated with the application being loaded, then the security domain will be invoked. However, if no security domain **310** is associated with the application which is being loaded, then the cryptographic key and the signature key of card domain **308** will be utilized. In contrast to the install commands sent to the smart card during the initialization phase, the post issuance install command is not issued in a secured environment, therefore it is preferably protected with a cryptographic key, such as a MAC/Signature key. Card domain **308** manages the post-issuance loading of a new application, while secure domain **310** ensures the validity and integrity of the new application once the new application has been loaded onto the smart card. If a secure domain **310** is not associated with the newly loaded application, then card domain **308** performs secure domain's **310** functions. Once the new application is post-issuance downloaded, various keys, such as an cryptographic key and a signature key, are preferably utilized for installation and personalization of the application.

A method and system for a smart card domain and a security domain has been disclosed. Software written according to the present invention may be stored in some form of computer-readable medium, such as memory or CD-ROM, or transmitted over a network, and executed by a processor.

Although the present invention has been described in accordance with the embodiment shown, one of ordinary skill in the art will readily recognize that there could be variations to the embodiment and these variations would be within the spirit and scope of the present invention. Accordingly, many modifications may be made by one of ordinary skill in the art without departing from the spirit and scope of the appended claims.

What is claimed is:

1. A smart card comprising:

a card life cycle having a plurality of states;

a memory including an indication of which of said states said card life cycle is in; and

a card domain application including

an issuer key associated with the issuer of said smart card,

a function for managing said life cycle of said smart card, and

a function for tracking the status of said life cycle of said smart card, whereby said card domain application represents the interests of the issuer and manages said card life cycle.

2. A smart card as recited in claim **1** wherein said card domain application further includes:

a function for blocking said smart card.

3. A smart card as recited in claim **1** wherein said states of said card life cycle include masked, initialized, load secured and blocked.

4. A smart card as recited in claim **1** wherein said states of said card life cycle are in an irreversible sequence and wherein said states of said card life cycle place said smart card into an increasing level of security.

5. A smart card as recited in claim **1** wherein the contents of said memory determines the state of said card life cycle.

6. A method of blocking a smart card comprising:

detecting a problem with said smart card by an application of said smart card;

sending a card block request from said application to a card domain application of said smart card, said card domain application having the capability to block said smart card;

determining by said card domain application whether said card block request is valid; and

blocking said smart card by said card domain application, whereby said smart card is not operational for a user.

7. A method as recited in claim **6** wherein said card domain application includes an issuer key associated with the issuer of said smart card, whereby said card domain application represents the interests of the issuer.

8. A method of moving a smart card through a sequence of card life cycle states, said method comprising:

receiving said smart card in a masked state, said masked state indicating that components necessary for initialization are available on said smart card;

initializing said smart card using an initialization key;

placing said smart card into an initialized state;

loading an application onto said smart card post-issuance; and

placing said smart card into a load secured state, whereby said smart card passes through a number of said states of said card life cycle.

9. A method as recited in claim **8** further comprising:

receiving a card block request;

blocking said smart card; and

placing said smart card into a blocked state, whereby said smart card is not operational for a user.

10. A method as recited in claim **8** wherein said card life cycle states are managed by a card domain application.

11. A method as recited in claim **8** wherein said states of said card life cycle are in an irreversible sequence.

12. A method as recited in claim **8** wherein said states of said card life cycle place said smart card into an increasing level of security.

13. A smart card comprising:

a first application having a sequence of life cycle states; and

a card domain application including

an issuer key associated with the issuer of said smart card,

a function for loading said application onto said smart card, said loading causing said first application to be placed into a loaded state,

a function for installing said application on said smart card, said installing causing said first application to be placed into an installed state, and

a function for registering said application on said smart card, said registering causing said first application to be placed into a registered state, whereby said card domain application represents the interests of the issuer and manages said first application.

14. A smart card as recited in claim 13 wherein said card domain application further includes:

a cryptographic service for loading said first application onto said smart card post-issuance.

15. A smart card as recited in claim 13 wherein said first application further includes:

a function for personalizing said first application, said personalizing causing said first application to be placed into a personalized state, whereby said personalizing is under the authority of said first application.

16. A smart card as recited in claim 13 wherein said first application further includes:

a function for blocking said first application, said blocking causing said first application to be placed into a blocked state, whereby said blocking is under the authority of said first application.

17. A method of moving an application of smart card through a sequence of application life cycle states, said method comprising:

receiving said application on said smart card, said receiving placing said application into a loaded state;

installing said application on said smart card, said installing placing said application into an installed state;

registering said application on said smart card, said registering placing said application into a registered state;

personalizing said application on said smart card, said personalizing placing said application into a personalized state, whereby said application is available for use.

18. A method as recited in claim 17 further comprising:

receiving an application block request;

blocking said application; and

placing said application into a blocked state, whereby said application is not available for use.

19. A method as recited in claim 17 further comprising:

receiving an application delete request;

deleting said application from said smart card; and

indicating said application is in a not available state, whereby said application is not available for use.

20. A method as recited in claim 17 wherein said application is received by being loaded into a memory of said smart card during initialization of said smart card, whereby said application is present on said smart card before issuance.

21. A method as recited in claim 17 wherein said application is received by being loaded onto said smart card post-issuance, whereby said application appears on said smart card after issuance.

22. A method of moving an application of smart card through a sequence of application life cycle states after issuance of said smart card, said method comprising:

issuing said smart card;

indicating within said smart card that said application is in a not available state;

loading said application onto said smart card post-issuance, said loading placing said application into a loaded state; and

installing said application on said smart card, said installing placing said application into an installed state, whereby said application is available for use on said smart card.

23. A method as recited in claim 22 further comprising:

personalizing said application on said smart card, said personalizing placing said application into a personalized state.

* * * * *

US006481632B2

(12) **United States Patent**
Wentker et al.

(10) **Patent No.: US 6,481,632 B2**
(45) **Date of Patent: *Nov. 19, 2002**

(54) **DELEGATED MANAGEMENT OF SMART CARD APPLICATIONS**

(75) Inventors: **David C. Wentker**, San Francisco, CA (US); **Klaus P. Gungl**, Sindelfingen (DE)

(73) Assignee: **Visa International Service Association**, San Francisco, CA (US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/427,517**

(22) Filed: **Oct. 26, 1999**

(65) **Prior Publication Data**

US 2002/0040936 A1 Apr. 11, 2002

Related U.S. Application Data

(60) Provisional application No. 60/121,810, filed on Feb. 25, 1999, provisional application No. 60/124,130, filed on Mar. 12, 1999, and provisional application No. 60/105,841, filed on Oct. 27, 1998.

(51) **Int. Cl.**[7] **G06K 19/06**

(52) **U.S. Cl.** **235/492**; 235/376; 235/380; 235/382; 235/487

(58) **Field of Search** 235/380, 492, 235/487, 376, 382

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,742,215 A 5/1988 Turpen et al. 235/487
4,831,245 A 5/1989 Igasawara

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

AT E100227 11/1994
DE 19607363 9/1996

(List continued on next page.)

OTHER PUBLICATIONS

Carol Hovenga Fancher, "In Your Pocket SmartCard", Feb. 1997, IEEE Specturm.

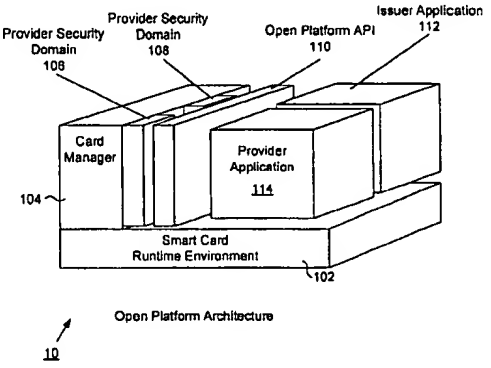
(List continued on next page.)

Primary Examiner—Karl D. Frech
Assistant Examiner—Seung Ho Lee
(74) *Attorney, Agent, or Firm*—Beyer Weaver & Thomas, LLP

(57) **ABSTRACT**

A smart card architecture includes a run-time environment, a card manager, one or more security domains, a provider application and an issuer application. One or more APIs provide communication. The life cycle of the card and card manager includes states: Pre-production, Ready, Initialized, Secured, Locked and Terminated. The life cycle of an application includes states: Installed, Selectable, Personalized, Blocked, Locked and Deleted. A card registry keeps track of card manager and application data elements. The functionality of a security domain on a smart card is extended to allow it to perform delegated management of smart card applications: delegated loading, installation and/or deletion of an application. A provider of an application is assured of more direct control and management of their application, yet an issuer still maintains some control over the management of the card. The card issuer empowers application providers to initiate changes to the issuer's smart cards that are pre-approved by the card issuer. A method of delegated loading of an application onto a smart card first receives a load command from an application provider via a card acceptance device. The load command includes an indication of an application to be loaded and an appended command authentication pattern. Next, the load command is verified using the command authentication pattern. Then, an application is received from an application provider via the card acceptance device; the application also includes an appended application authentication pattern which is used to verify the application. Finally, the application is loaded into memory of the smart card.

16 Claims, 14 Drawing Sheets



U.S. PATENT DOCUMENTS

5,332,889	A		7/1994	Lundstrom et al. 235/380
5,378,884	A		1/1995	Lundstrom et al. 235/441
5,530,232	A		6/1996	Taylor 235/380
5,578,808	A	*	11/1996	Taylor 235/380
5,583,933	A		12/1996	Mark 379/355
5,901,303	A		5/1999	Chew 395/400
5,923,884	A	*	7/1999	Pryret et al. 395/712
6,005,942	A	*	12/1999	Chan et al. 235/380
6,164,549	A	*	12/2000	Richards 235/492
6,167,521	A	*	12/2000	Smith et al. 713/200

FOREIGN PATENT DOCUMENTS

EP	0193635	9/1986
EP	0658862	6/1995
EP	0795844	9/1997
EP	0798673	10/1997
WO	98/43212	10/1998

OTHER PUBLICATIONS

Chaum et al., "SmartCard 2000: The Future of IC Cards", Oct. 19, 1987, Elsevier Science Publishers, B.V.

Steven Levy, "E-Money (That's What I Want)", Dec. 1994, Wired Magazine.

Carol H. Fancher, "Smart Cards as Potential Applications Grow, Computers in the Wallet are Making Unobstrusive Inroads", Aug. 1996, Scientific American Website.

Jerome Svigals, "Smart Cards The New Bank Cards", 1985, MacMillan Publishing Company.

Roy Bright, "SmartCards: Principles, Practice, Applications", 1998, Ellis Horwood Limited.

Jerome Svigals, "SmartCards The Ultimate Personal Computer", 1985, MacMillan Publishing Company.

Hawkes et al., "Integrated Circuit Cards, Tags and Tokens", 1990, BSP Professional Books.

David Naccache, "Cryptographic Smart Cards", Jun. 3, 1996, IEEE Micro 1996 Website.

Zoreda et al., "Smart Cards", 1994, Artech House.

"Identification Card Systems—Inter-Sector Electronic Purse Part I: Concepts and Structures", 1994, European Standard, prEN 1546.

"Identification Card Systems—Inter-Sector Electronic Purse Part 2: Security Architecture", 1994, European Standard, prEN XXXXX-2.

"Identification Card System—Inter-Sector Electronic Purse Part 3: Data Elements and Interchanges", 1994, European Prestandard, prEN 1546-3.

"Identification Card System—Inter-Sector Electronic Purse Part 4: Devices", 1994, European Prestandard, prEN 1546-4.

"Identification Cards—Integrated Circuit(s) Cards With Contacts Part 1: Physical Characteristics", 1987, International Standard, ISO 7816-1, First Edition.

"Identification Cards—Integrated Circuit(s) Cards With Contacts Part 2: Dimensions and Location of the Contacts", 1988, International Standard, ISO 7816-2, First Edition.

"Identification Cards—Integrated Circuit(s) Cards With Contacts Part 3: Electronic Signals and Transmission Protocols", International Standard, ISO/IEC 7816-3, First Edition.

"Identification Cards—Integrated Circuit(s) Cards with Contacts Part 4: Inter-Industry Commands for Interchange", International Standard, ISO/IEC 7816-4, First Edition.

"Identification Cards—Integrated Circuit(s) Cards With Contacts Part 5: Numbering System and Registration Procedure for Application Identifiers", 1993, International Standard, ISO/IEC DIS 7816-5.

"International Cards—Integrated Circuit(s) Cards With Contacts Part 6: Inter-Industry Data Elements", 1995, International Standard, ISO/IEC DIS 7816-6.

"Bank Cards—Magnetic Stripe Data Content For Track 3", 1987, International Standard, ISO 4909 Second Edition.

"Identification Cards—Physical Characteristics", 1995, International Standard, ISO/IEC 7810, Second Edition.

"Identification Cards—Recording Technique—Part 1: Embossing", 1995, International Standard, ISO/IEC 7811-1, Second Edition.

"Identification Cars—Recording Technique—Part 2: Magnetic Strip", 1995, International Standard, ISO/IEC 7811-2, Second Edition.

"Identification Cards—Recording Technique—Part 3: Location of Embossed Characters on ID-1 Cards", 1995, International Standard, ISO/IEC 7811-5, Second Edition.

"Identification Cards—Recording Technique—Part 4: Location of Read-Only Magnetic Tracks—Tracks 1 & 2", 1995, International Standard, ISO/IEC 7811-4, Second Edition.

"Identification Cards—Recording Technique—Part 5: Location of Read-Write Magnetic Track—Track 3", International Standard, ISO/IEC 7811-5, Second Edition.

"Identification Cards—Recording Technique—Part 6: Magnetic Stripe-High Coercivity", 1996, International Standard, ISO/IEC 7811-6, First Edition.

"Identification Cards—Financial Transaction Cards", 1990, International Standard, ISO/IEC 7813, Third Edition.

"Identification Cards—Financial Transaction Cards Amendment 1" 1996, International Standard, ISO/IEC 7813, Fourth Edition.

"Identification Cards—Contactless Integrated Circuit(s) Cards—Part 1: Physical Characteristics", 1992, International Standard, ISO/IEC 10536-1, First Edition.

"Identification Cards—Contactless Integrated Circuit(s) Cards—Part 2: Dimensions and Location of Coupling Areas", 1995, International Standard, ISO/IEC 10536-2, First Edition.

"Identification Cards—Contactless Integrated Circuit(s) Cards—Part 3: Electronic Signals and Reset Procedures", 1996, International Standard, ISO/IEC 10536-3, First Edition.

"Financial Transaction Cards—Security Architecture of Financial Transaction System Using Integrated Circuit Cards—Part 1: Card Life Cycle", Sep. 15, 1991, International Standard, ISO/IEC 10202-1, First Edition.

Hiro Shogase, "The Very Smart Card: A Plastic Pocket Bank", IEEE Sepctrum, Oct. 1988.

* cited by examiner

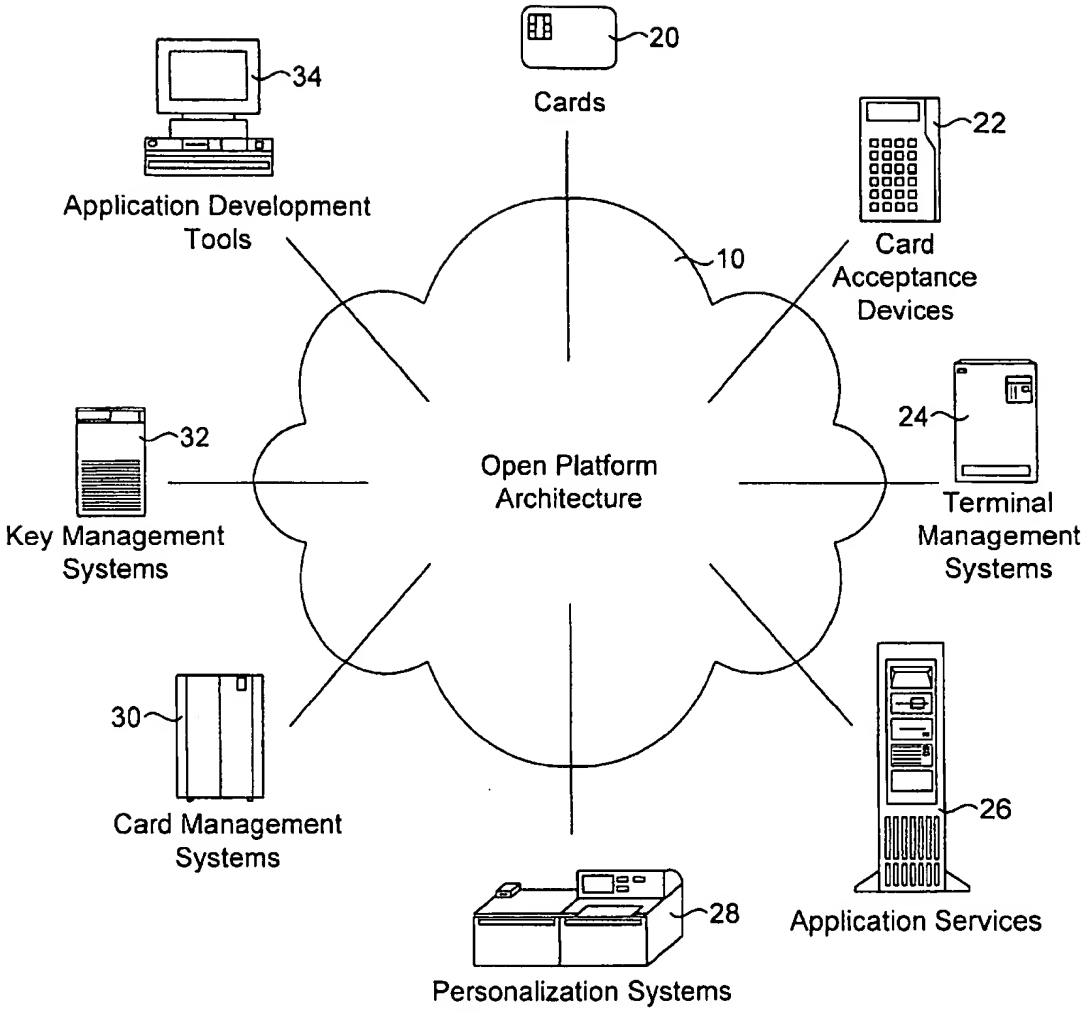

20
Cards
34
Application Development Tools
22
Card Acceptance Devices
10
Open Platform Architecture
24
Terminal Management Systems
32
Key Management Systems
30
Card Management Systems
26
Application Services
28
Personalization Systems

FIG. 1

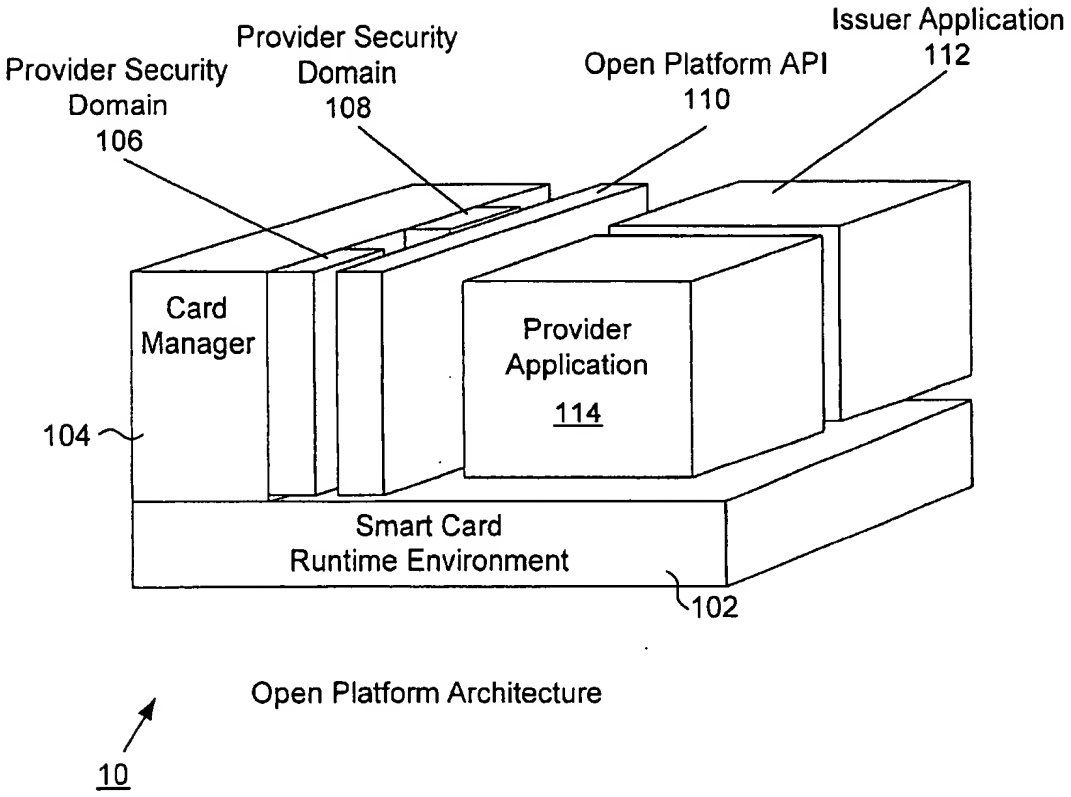

Provider Security Domain 106
Provider Security Domain 108
Open Platform API 110
Issuer Application 112
Card Manager
104
Provider Application 114
Smart Card Runtime Environment
102
Open Platform Architecture
10

FIG. 2

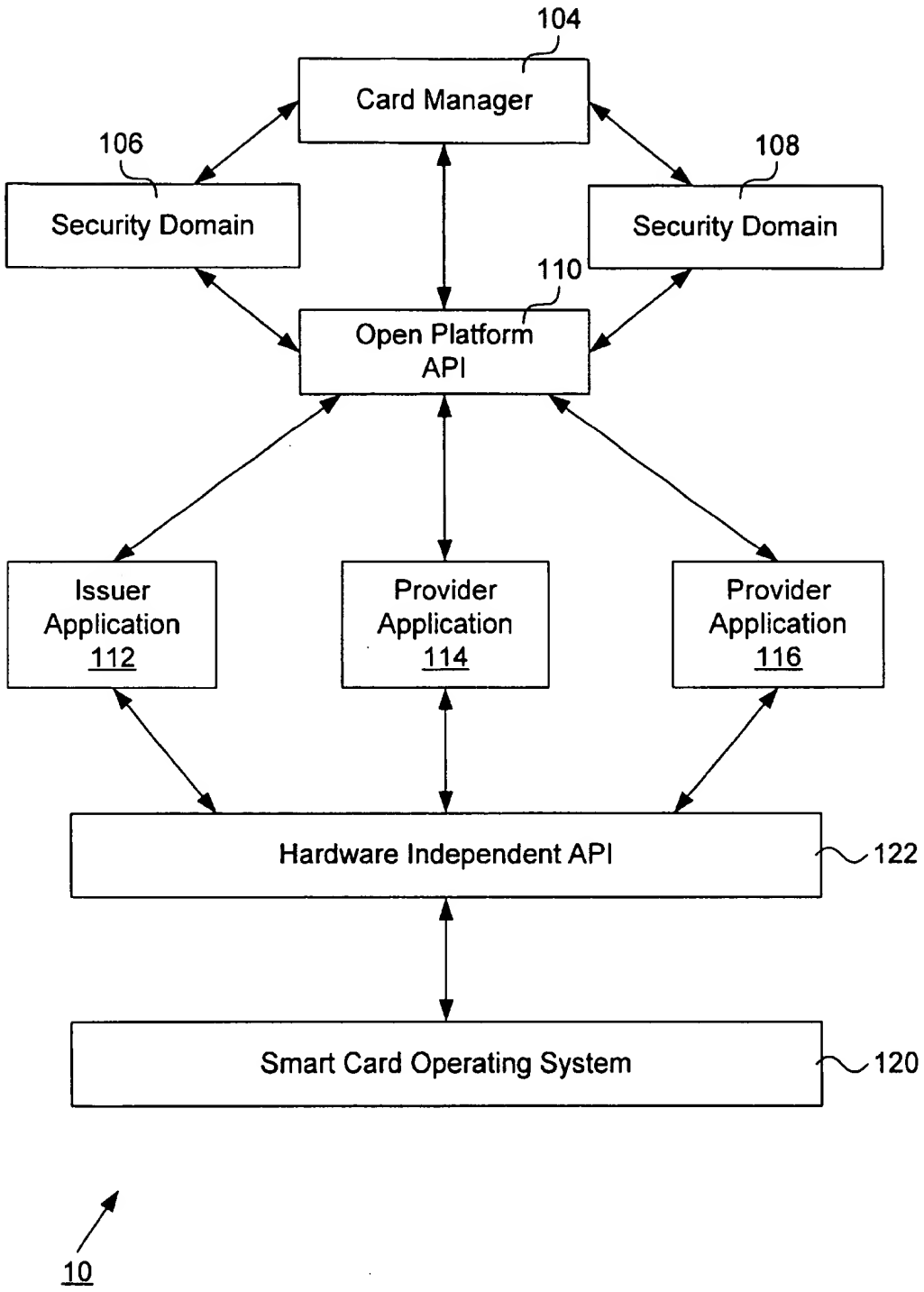

104
Card Manager
106
108
Security Domain
Security Domain
110
Open Platform API
Issuer Application 112
Provider Application 114
Provider Application 116
Hardware Independent API
122
Smart Card Operating System
120
10

FIG. 3

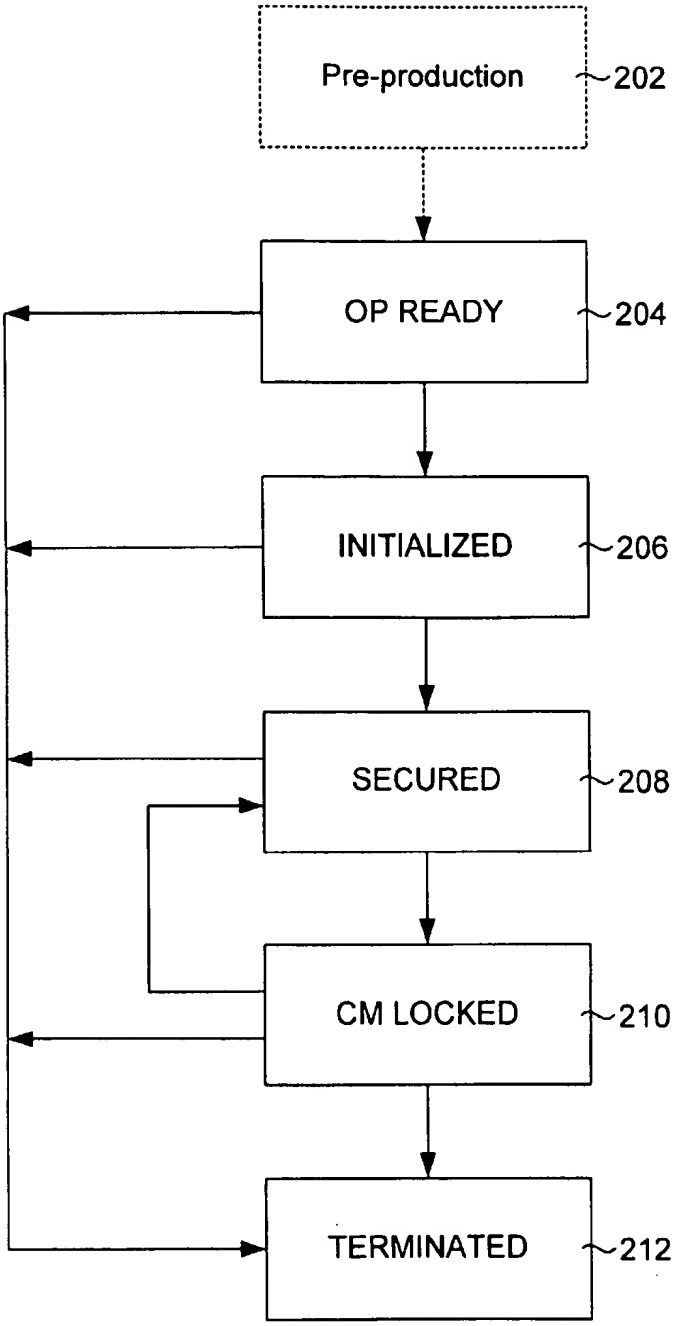
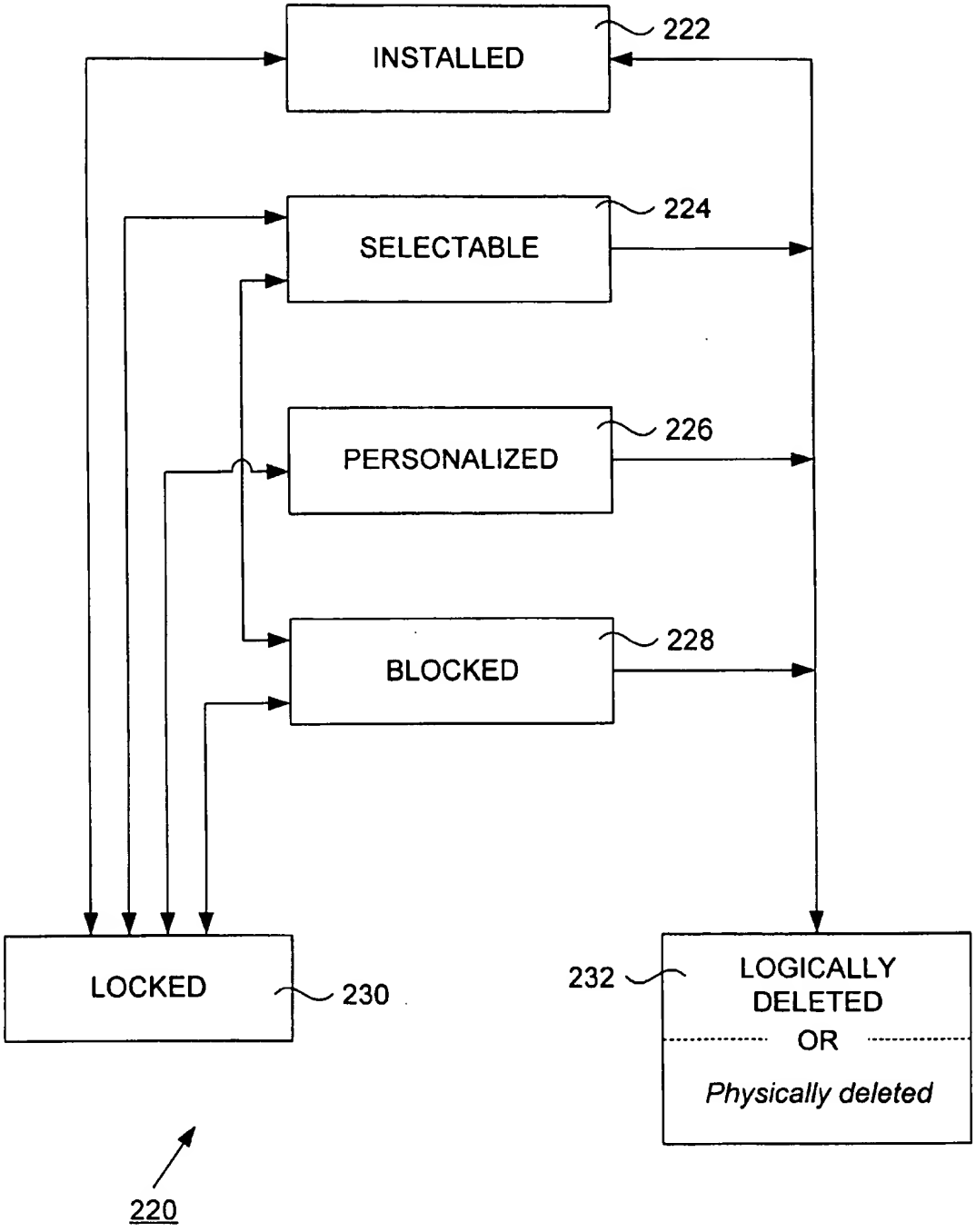

Pre-production
202
OP READY
204
INITIALIZED
206
SECURED
208
CM LOCKED
210
TERMINATED
212
200
Card Manager Life Cycle

FIG. 4



Application Life Cycle

FIG. 5

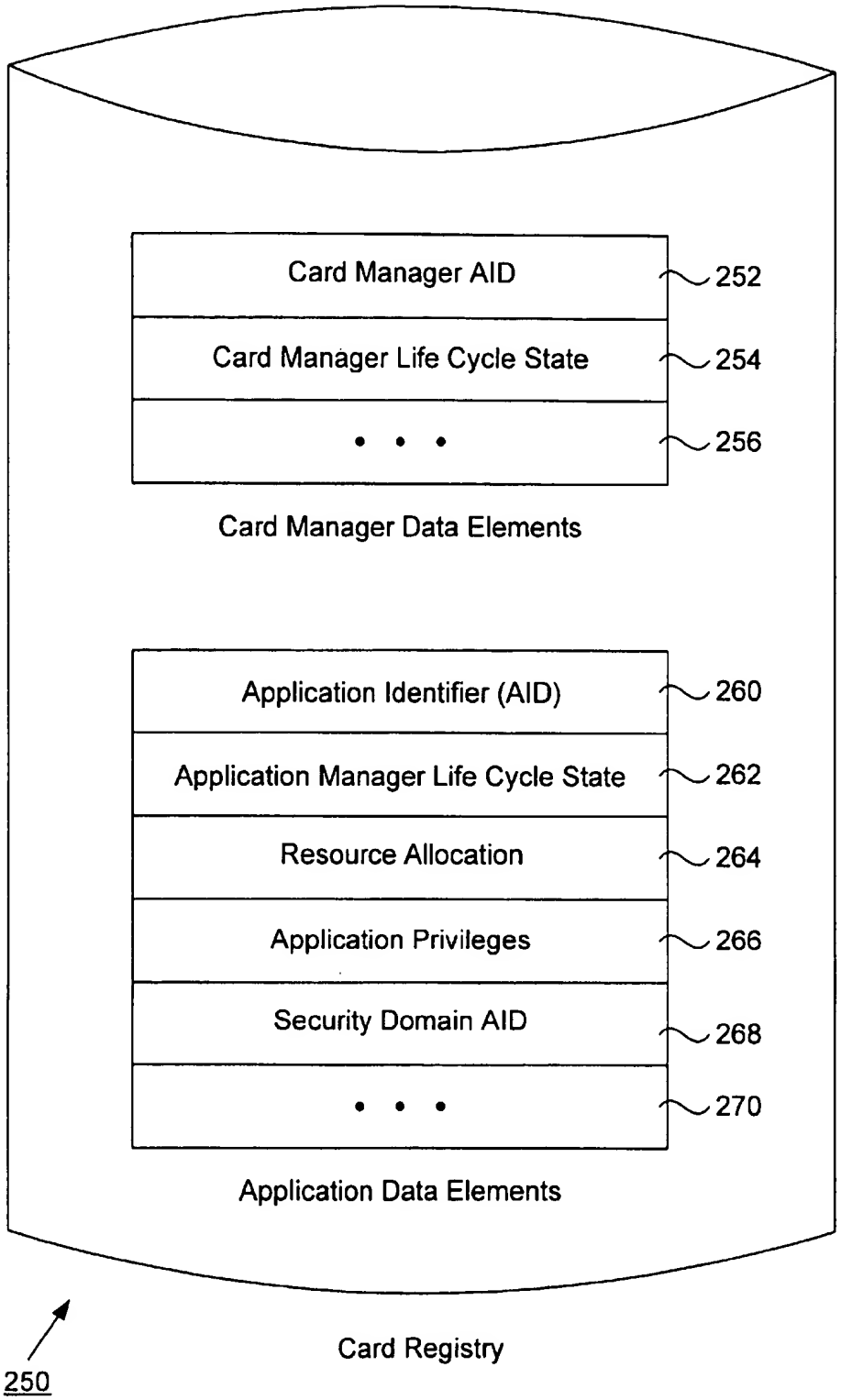

Card Manager AID
252
Card Manager Life Cycle State
254
• • •
256
Card Manager Data Elements
Application Identifier (AID)
260
Application Manager Life Cycle State
262
Resource Allocation
264
Application Privileges
266
Security Domain AID
268
• • •
270
Application Data Elements
Card Registry
250

FIG. 6

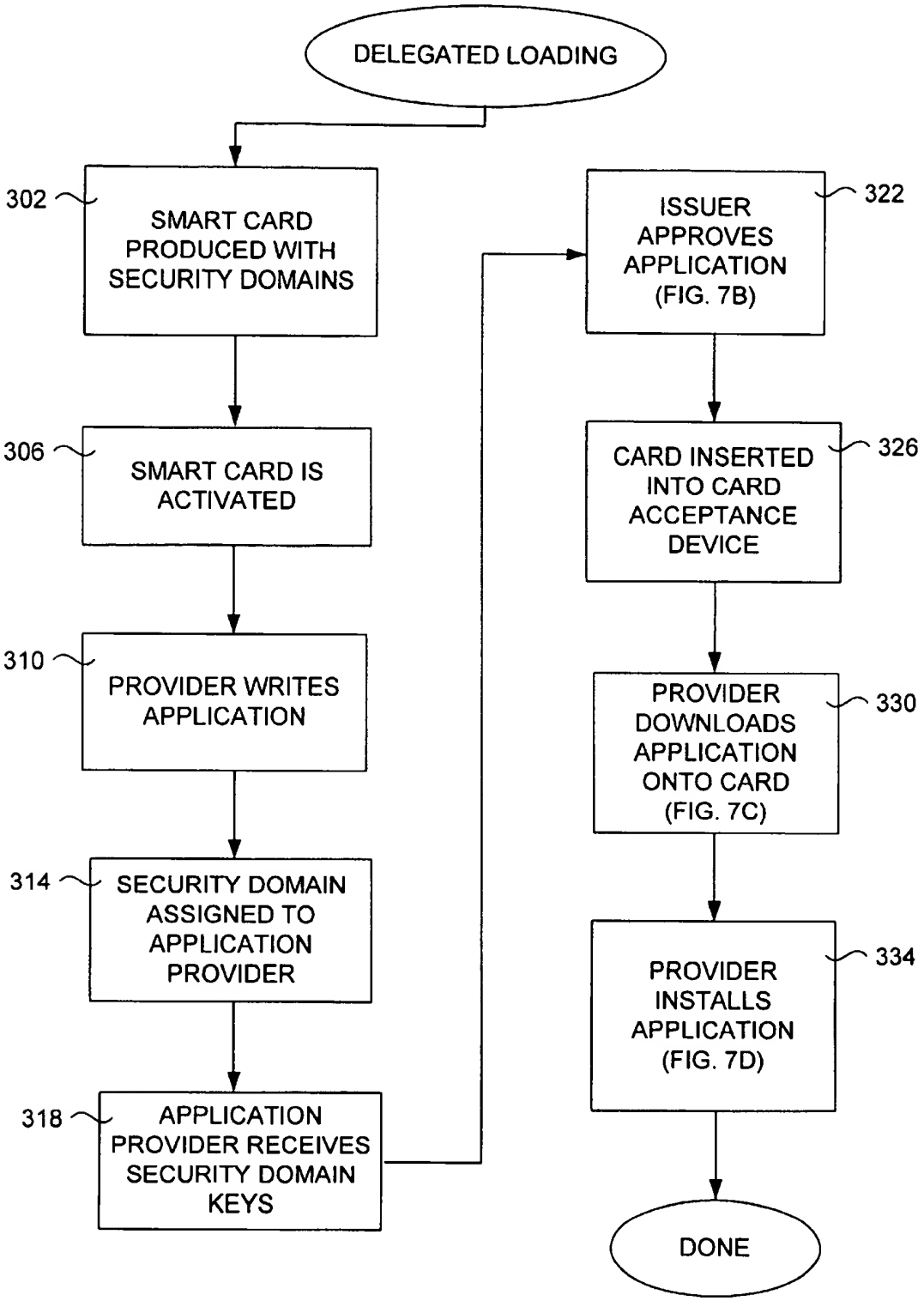

DELEGATED LOADING
302
SMART CARD PRODUCED WITH SECURITY DOMAINS
306
SMART CARD IS ACTIVATED
310
PROVIDER WRITES APPLICATION
314
SECURITY DOMAIN ASSIGNED TO APPLICATION PROVIDER
318
APPLICATION PROVIDER RECEIVES SECURITY DOMAIN KEYS
322
ISSUER APPROVES APPLICATION (FIG. 7B)
326
CARD INSERTED INTO CARD ACCEPTANCE DEVICE
330
PROVIDER DOWNLOADS APPLICATION ONTO CARD (FIG. 7C)
334
PROVIDER INSTALLS APPLICATION (FIG. 7D)
DONE

FIG. 7A

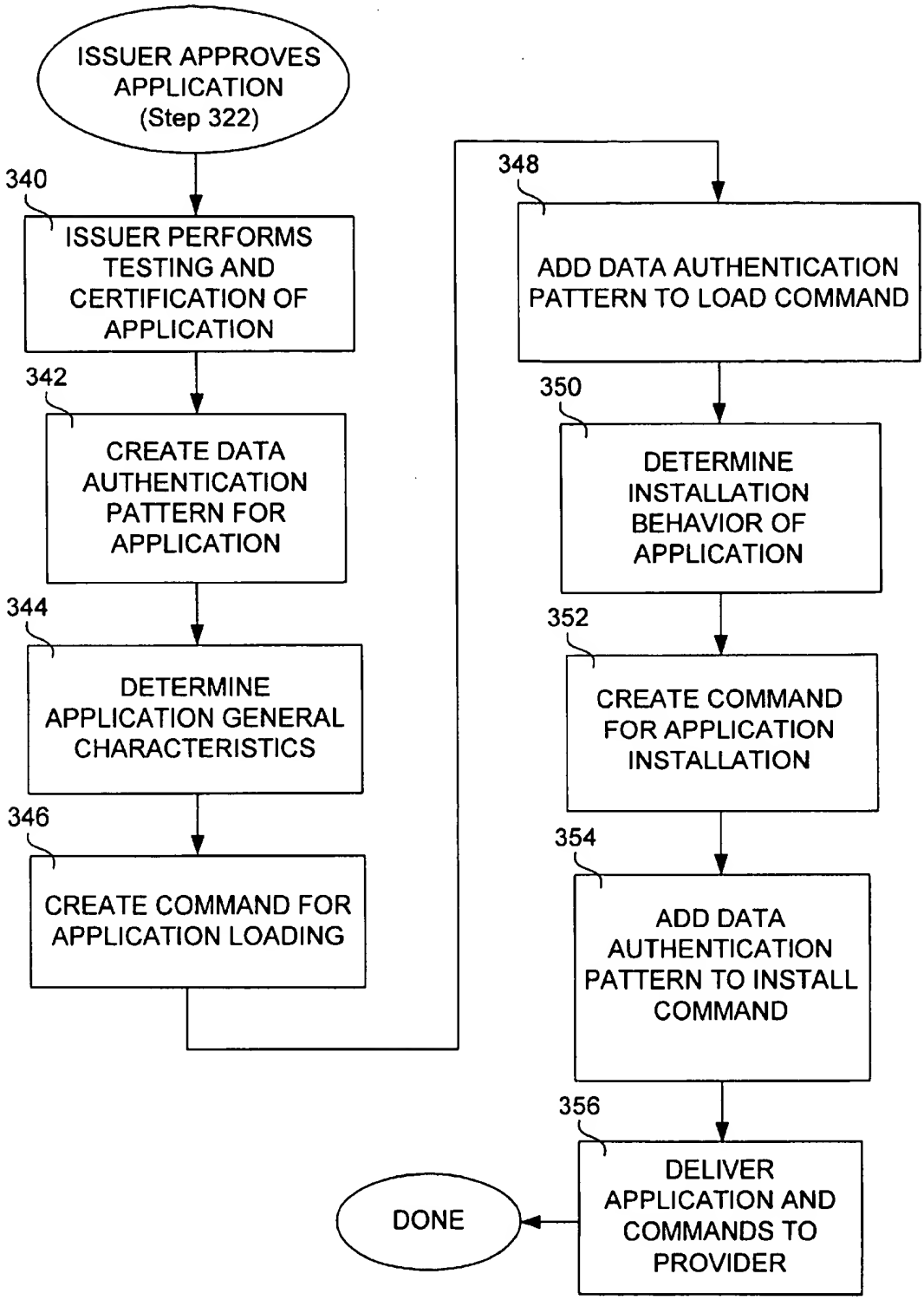

ISSUER APPROVES APPLICATION (Step 322)
340
ISSUER PERFORMS TESTING AND CERTIFICATION OF APPLICATION
342
CREATE DATA AUTHENTICATION PATTERN FOR APPLICATION
344
DETERMINE APPLICATION GENERAL CHARACTERISTICS
346
CREATE COMMAND FOR APPLICATION LOADING
348
ADD DATA AUTHENTICATION PATTERN TO LOAD COMMAND
350
DETERMINE INSTALLATION BEHAVIOR OF APPLICATION
352
CREATE COMMAND FOR APPLICATION INSTALLATION
354
ADD DATA AUTHENTICATION PATTERN TO INSTALL COMMAND
356
DELIVER APPLICATION AND COMMANDS TO PROVIDER
DONE

FIG. 7B

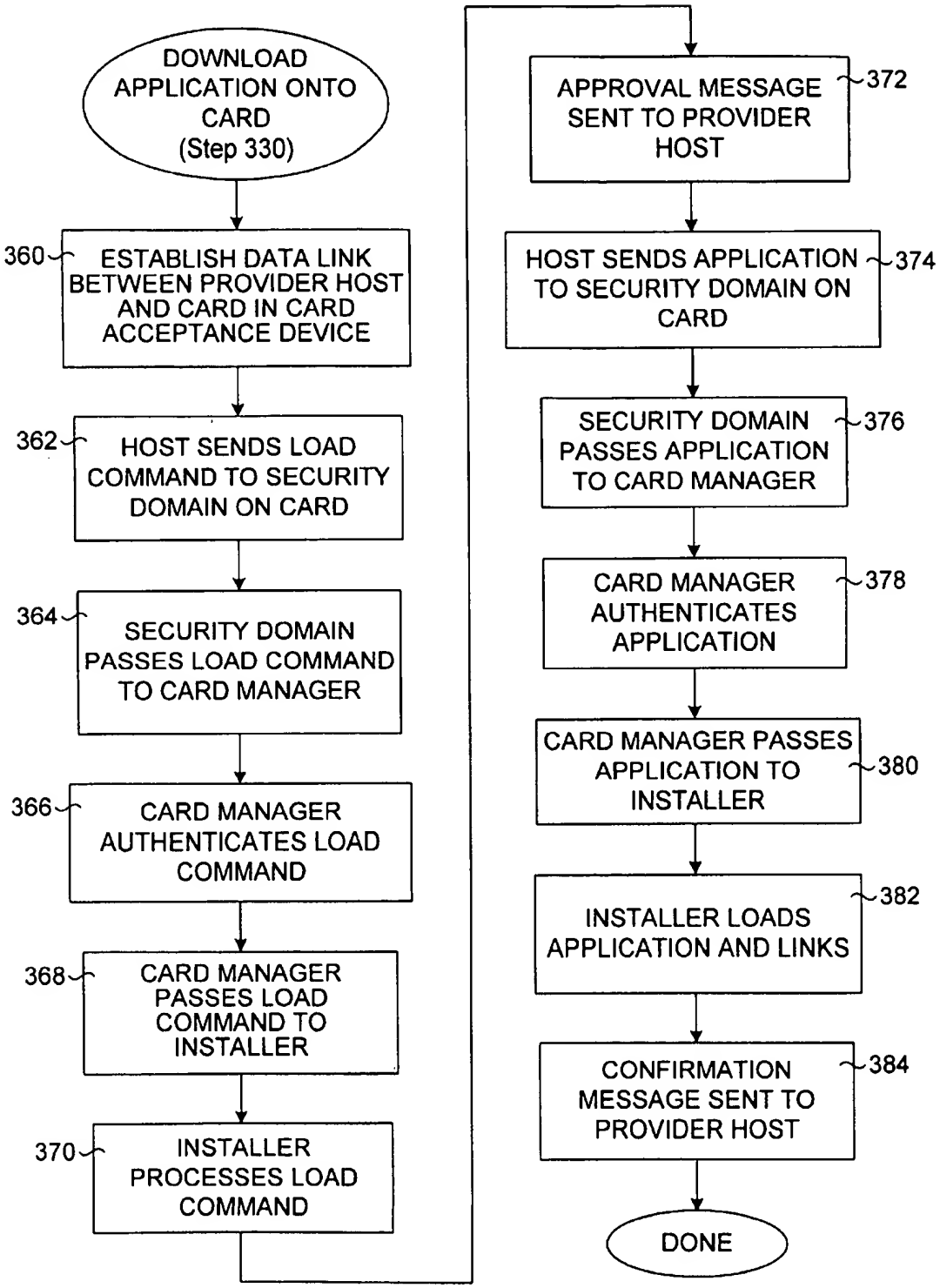

DOWNLOAD APPLICATION ONTO CARD (Step 330)
360
ESTABLISH DATA LINK BETWEEN PROVIDER HOST AND CARD IN CARD ACCEPTANCE DEVICE
362
HOST SENDS LOAD COMMAND TO SECURITY DOMAIN ON CARD
364
SECURITY DOMAIN PASSES LOAD COMMAND TO CARD MANAGER
366
CARD MANAGER AUTHENTICATES LOAD COMMAND
368
CARD MANAGER PASSES LOAD COMMAND TO INSTALLER
370
INSTALLER PROCESSES LOAD COMMAND
372
APPROVAL MESSAGE SENT TO PROVIDER HOST
374
HOST SENDS APPLICATION TO SECURITY DOMAIN ON CARD
376
SECURITY DOMAIN PASSES APPLICATION TO CARD MANAGER
378
CARD MANAGER AUTHENTICATES APPLICATION
380
CARD MANAGER PASSES APPLICATION TO INSTALLER
382
INSTALLER LOADS APPLICATION AND LINKS
384
CONFIRMATION MESSAGE SENT TO PROVIDER HOST
DONE

FIG. 7C

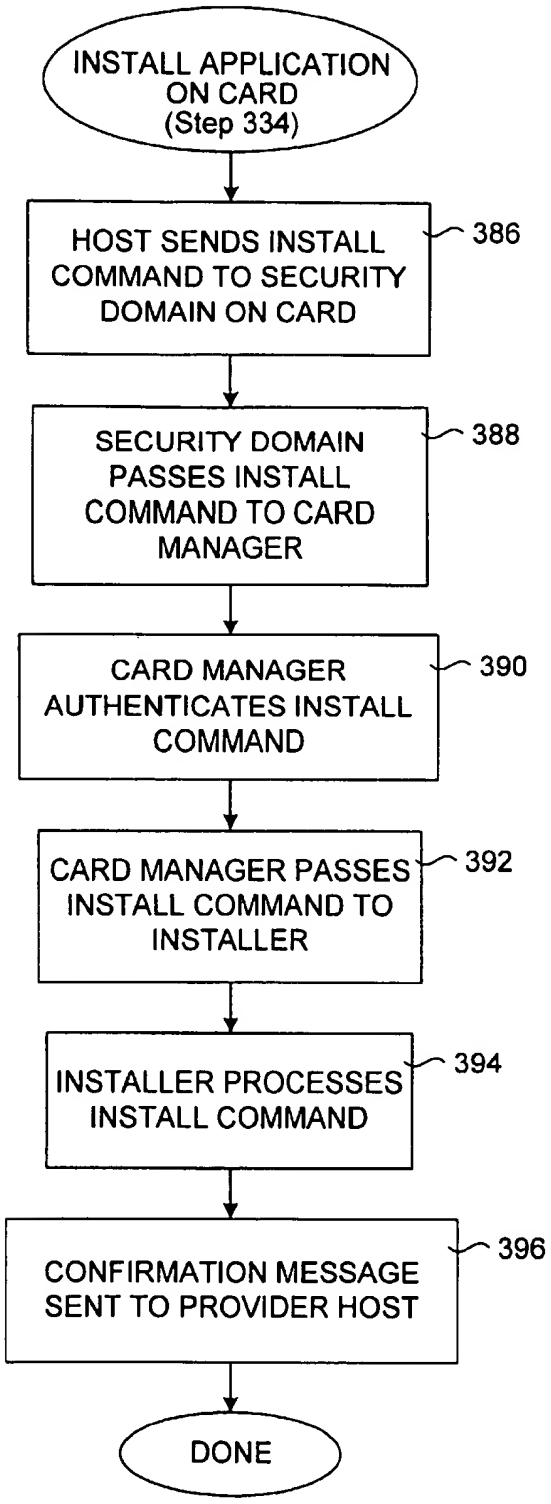

INSTALL APPLICATION ON CARD (Step 334)
HOST SENDS INSTALL COMMAND TO SECURITY DOMAIN ON CARD
386
SECURITY DOMAIN PASSES INSTALL COMMAND TO CARD MANAGER
388
CARD MANAGER AUTHENTICATES INSTALL COMMAND
390
CARD MANAGER PASSES INSTALL COMMAND TO INSTALLER
392
INSTALLER PROCESSES INSTALL COMMAND
394
CONFIRMATION MESSAGE SENT TO PROVIDER HOST
396
DONE

FIG. 7D

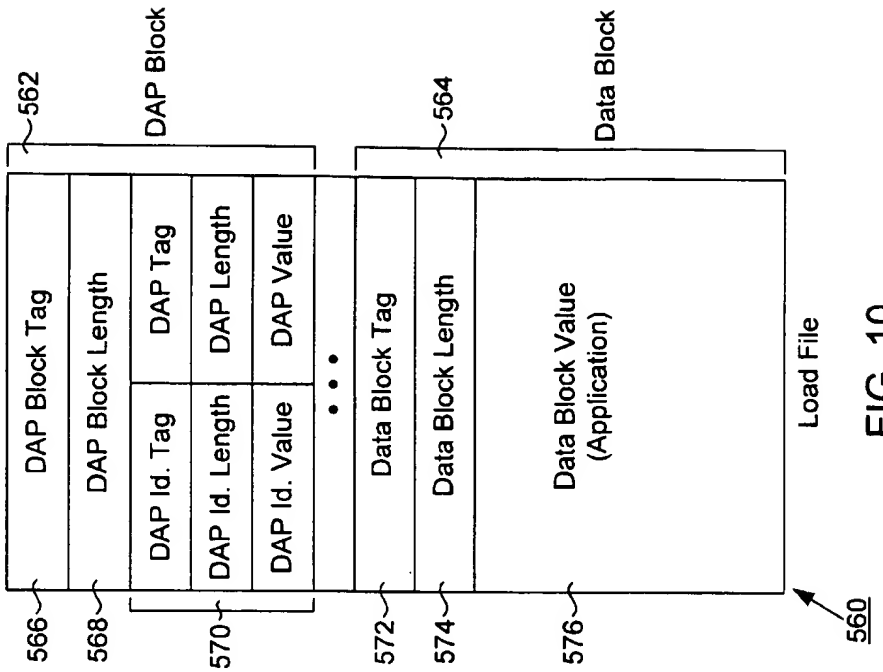


FIG. 10

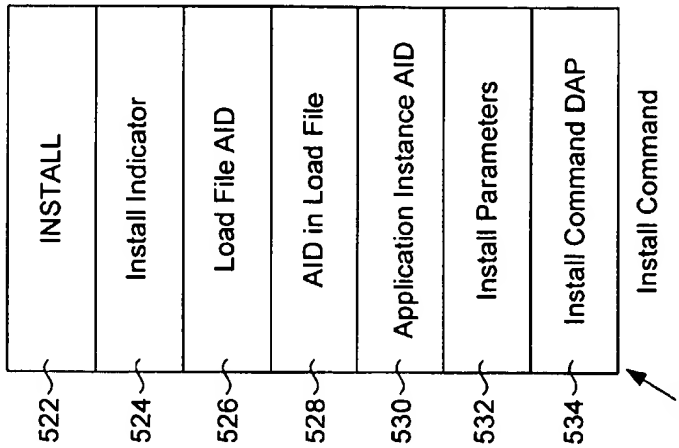


FIG. 9

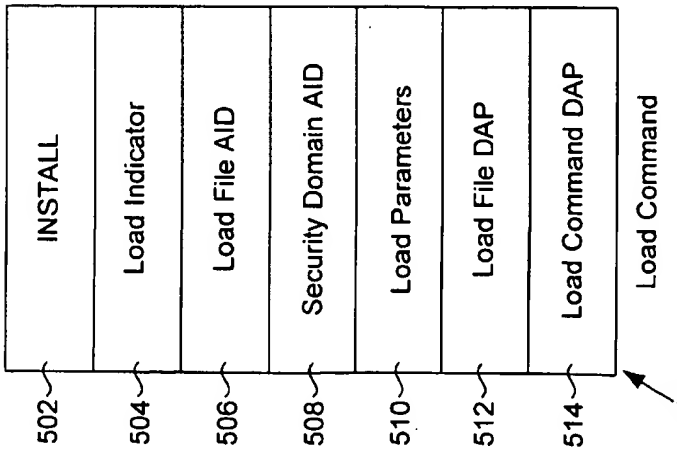


FIG. 8

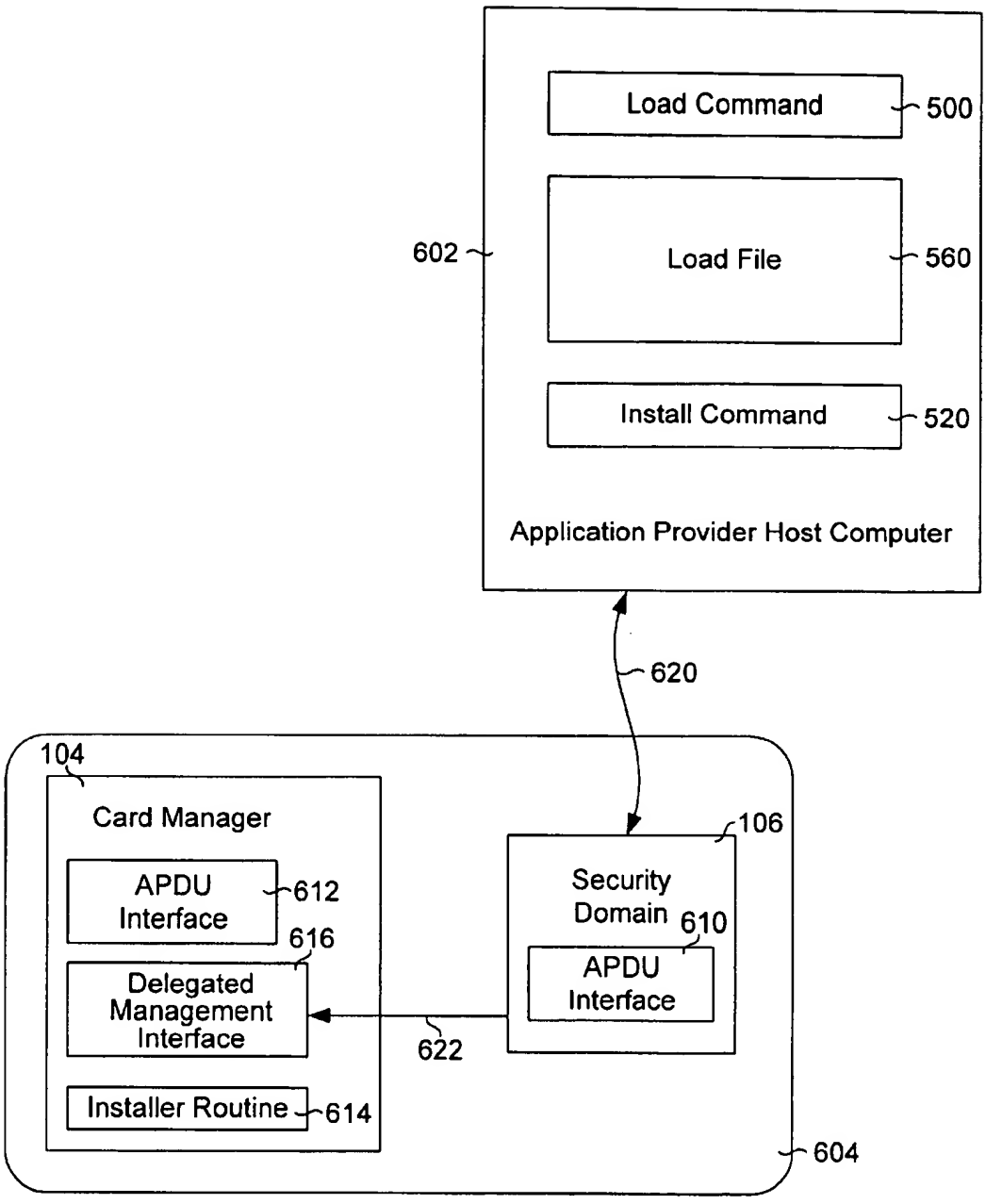

Load Command
500
602
Load File
560
Install Command
520
Application Provider Host Computer
620
104
Card Manager
APDU Interface
612
616
Delegated Management Interface
Installer Routine
614
622
106
Security Domain
610
APDU Interface
604

FIG. 11

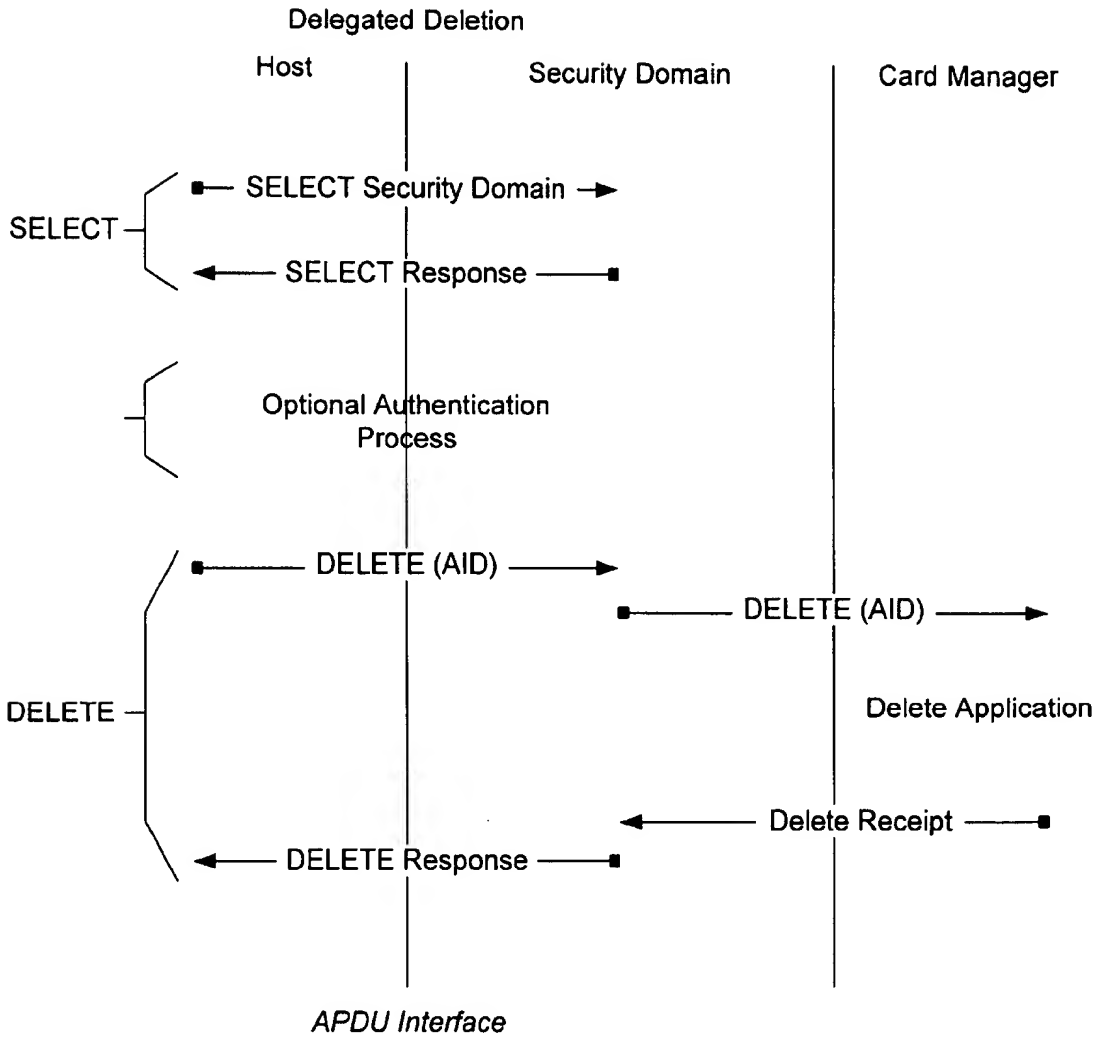

Delegated Deletion
Host
Security Domain
Card Manager
SELECT Security Domain
SELECT
SELECT Response
Optional Authentication Process
DELETE (AID)
DELETE (AID)
DELETE
Delete Application
Delete Receipt
DELETE Response
APDU Interface

FIG. 12

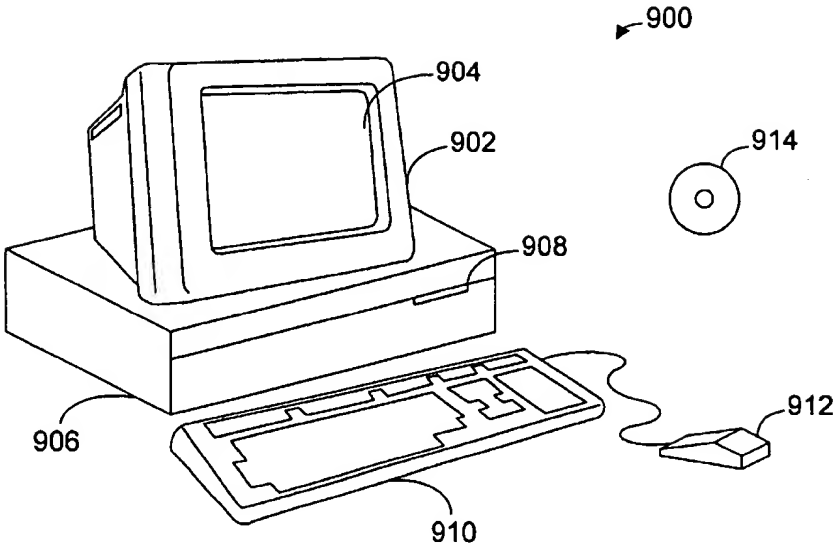

900
904
902
914
908
912
906
910

FIG. 13

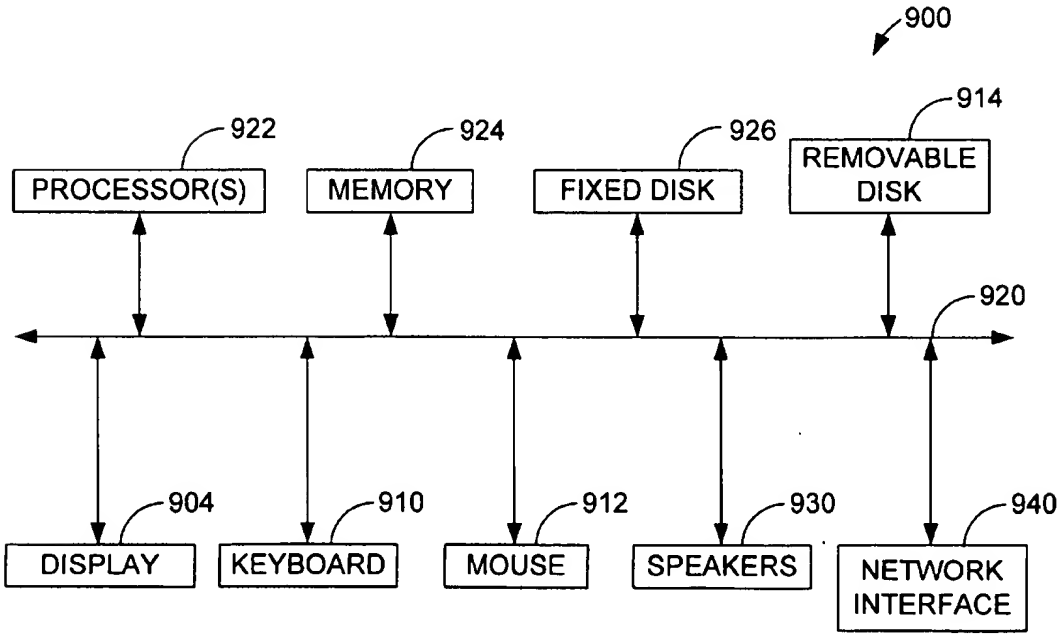

900
922
924
926
914
PROCESSOR(S)
MEMORY
FIXED DISK
REMOVABLE DISK
920
904
910
912
930
940
DISPLAY
KEYBOARD
MOUSE
SPEAKERS
NETWORK INTERFACE

FIG. 14

DELEGATED MANAGEMENT OF SMART CARD APPLICATIONS

This application claims priority of U.S. provisional patent application Nos. 60/105,841, 60/121,810 and 60/124,130 filed Oct. 27, 1998, Feb. 25, 1999 and Mar. 12, 1999 respectively, each entitled "Visa Open Platform Card Specification," which are hereby incorporated by reference.

This application is also related to U.S. patent application Ser. Nos. 09/046,993 and 09/046,994.

FIELD OF THE INVENTION

The present invention relates generally to smart cards. More specifically, the present invention relates to a technique for delegating the management of applications on a smart card such as loading, installation and deletion.

BACKGROUND OF THE INVENTION

Smart card technologies hold great promise as the replacement for magnetic stripe card technology. The adoption of smart cards, however, on a massive scale has been slow to develop. One reason for this slow adoption is the lack of standards among the many different vendor implementations of smart cards and the difficulties with implementing a new technology.

Recently, significant standards in the smart card area have been created. The standards, however, have been primarily targeted at either low levels of interoperability, such as the mechanical and electrical standards specified in the EMV specifications, or at the application layer in terms of developing standard chip credit, debit and purse applications. The main benefit of the standards has been realized in single-application smart cards, but has not significantly improved the situation for multi-applications smart cards.

The mid-1990s saw the introduction of various open systems standards for application development. For example, three technologies in this area are JAVA Card from Sun Microsystems, Inc., Smart Card for Windows from Microsoft Corporation, and MULTOS from MAOSCO, Ltd. These technology standards provide an important part of the solution toward common programming standards allowing application portability between different manufacturers card implementations. Other recent efforts have also addressed particular issues with multi-application smart cards. For example, U.S. patent application Ser. No. 09/046,994 filed Mar. 24, 1998, and U.S. patent application Ser. No. 09/046,993 filed Mar. 4, 1998 address issues related to post-issuance downloading and life cycle, each of which are hereby incorporated by reference.

In prior art smart cards only the issuer of the card has been allowed to perform certain management functions of applications such as loading an application onto the card, installing the application and deleting the application from the card. This reliance upon the issuer exclusively for loading, installing and deleting applications can lead to some difficulties. For example, should a store develop a loyalty application for its customers that it wishes to load and install onto their customer's smart cards, the store would be precluded from doing so if only the card issuer is allowed to perform such functions. Arranging for the store to contact the issuer, and arranging for the issuer to load the store's application onto its customer's smart cards presents basic logistical problems such as application security and access to cards. For example, it would be best if the store could download an application onto a customer card while the customer visited the store. If only the issuer can download the application, it becomes more difficult to access the customer card.

In addition to logistical problems, there are privacy issues with regard to an application developer's private customer data. For example, if a loyalty application of a particular third party is loaded onto a smart card by the issuer, it may be still necessary to add private customer-specific data onto each individual card for use by loyalty application. This private customer information may very well be the private property of the third party, and the third party may be unwilling to transfer this private information to a card issuer to allow the card issuer to load and install the third party's loyalty data. For instance, Hertz would likely be unwilling to provide private customer information regarding it top renters to the bank that is loading Hertz's application onto a smart card.

Other mechanical difficulties are presented should a customer desire to download and install a third party's application at the third party's site if only the issuer is allowed to download and install an application. For example, should a customer wish to download a loyalty application while at the third party's place of business during a smart card transaction, it would be first be necessary for the card acceptance device to connect to the issuer host to download the loyalty application, and then connect to the third party's host computer in order to receive custom information for the initialization and personalization of that application. Such multiple connections at load time make the transaction more complex, time consuming and are more prone to failure. In addition, as a practical matter, should a customer wish to download and install an application onto his smart card, it is more than likely that the customer is physically present at a third party site rather than at the issuer's site.

Further difficulties may be encountered if only the issuer is allowed to delete an application that belongs to the application provider/developer from a customer smart card. For example, the application is the responsibility of the application provider and is his liability. Should an agreement expire between the provider and the issuer, it may be more desirable for the provider to be able to delete his property (the application) from the smart card, rather than relying upon the issuer to do so for him. A further difficulty encountered if only the issuer is allowed to delete an application relates to the application data. When an application is deleted, the application data is deleted as well. Therefore, it would be desirable to allow the application provider to extract any relevant data (e.g., loyalty points from a loyalty application) from the card before the application is deleted. Since the card issuer does not have the provider's application keys, it would be near impossible for the card issuer to extract any information that required any kind of authentication. (The provider may also not desire that the issuer have access to the extracted information.)

Therefore, it would be desirable to allow another entity besides the card issuer to manage various functions associated with card applications such as loading, installing and deleting. It would further be desirable to allow the issuer to still be able to manage and control which applications are present on the smart card.

SUMMARY OF THE INVENTION

To achieve the foregoing, and in accordance with the purpose of the present invention, a technique is disclosed that extends the functionality of a security domain and allows it to perform delegated management of smart card applications. For example, embodiments of the present invention allow a security domain to perform delegated loading, installation and/or deletion of an application. By

delegating this management to their security domain, a provider of an application is assured of more direct control and management of their application, yet an issuer still maintains some control over the management of the card.

This concept of delegated management allows the card issuer the option of empowering application providers to initiate changes to the issuer's smart cards that are pre-approved by the card issuer. This pre-approval ensures that only card content changes that the card issuer has approved will be accepted and processed by a card manager of a smart card. This delegation of control in the card update process allows application providers more flexibility in managing their own applications on the card issuer's cards.

In one embodiment, a method of delegated loading of an application onto a smart card first receives a load command from an application provider via a card acceptance device. The load command includes an indication of an application to be loaded and an appended command authentication pattern. Next, the load command is verified using the command authentication pattern. Then, an application is received from an application provider via the card acceptance device; the application also includes an appended application authentication pattern which is used to verify the application. Finally, the application is loaded into memory of the smart card. Thus, an application provider is allowed to load an application onto a smart card.

In another embodiment, a system for delegated loading of an application onto a smart card includes a host computer under control of an application provider and a software application to be loaded onto a smart card. The application includes an appended application authentication pattern produced by an issuer of the smart card that verifies the application to the smart card. The system also includes a smart card acceptance device linked to the host computer and a smart card included in the card acceptance device. The smart card includes code arranged to verify the application using the application authentication pattern. Thus, the application provider is allowed to load the application onto the smart card.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention, together with further advantages thereof, may best be understood by reference to the following description taken in conjunction with the accompanying drawings in which:

FIG. **1** illustrates symbolically and environment in which the Open Platform architecture provides benefits for smart card holders, issuers, application developers and other entities.

FIG. **2** illustrates in further detail the Open Platform architecture as it may be implemented upon a smart card.

FIG. **3** is another illustration of the Open Platform architecture of FIG. **2** suitable for explaining the present invention.

FIG. **4** illustrates the card manager life cycle state transitions according to one embodiment of the invention.

FIG. **5** illustrates application life cycle state transitions according to one embodiment of the invention.

FIG. **6** illustrate a card registry database according to one embodiment of the invention.

FIG. **7**A is a flow diagram describing a technique for performing delegated loading.

FIG. **7**B is a flow diagram that describes how an issuer approves an application for delegated loading and installation.

FIG. **7**C is a flow diagram describing one embodiment of the download application step of FIG. **7**A.

FIG. **7**D is a flow diagram describing the install application step of FIG. **7**A.

FIGS. **8** and **9** illustrate examples of a load and an install command that may be created in steps **346** and **352** of FIG. **7**B.

FIG. **10** illustrates a load file containing an application according to one embodiment of the invention.

FIG. **11** illustrates an embodiment in which an application may be downloaded from an application provider host computer to a smart card in a delegated manner.

FIG. **12** is a flow diagram describing a technique for performing delegated deletion.

FIGS. **13** and **14** illustrate a computer system suitable for implementing embodiments of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The present invention is suitable for use with either single or multi-application smart cards. A multi-application smart card may come from a variety of manufacturers and may use any of a number of operating systems. By way of example, a smart card may use the JAVA Card operating system or the Smart Card for Windows operating system. As used herein, "smart card" refers to any of these single-application or multi-application smart cards. In one particular embodiment, the present invention works well with the "Open Platform" architecture as defined in *Open Platform Card Specification Version* 2.0, Apr. 19, 1999, available from Visa International Service Association. This architecture in one embodiment is based upon the JAVA Card operating system and provides a hardware-neutral, vendor-neutral, application-independent card-management standard. The standard provides a common security and card management architecture and defines a flexible and powerful standard for card issuers to create multi-application smart cards.

Smart Cards

The present invention is applicable to smart cards. Also termed chip cards, integrated circuit cards, memory cards or processor cards, a smart card is typically a credit card-sized plastic card that includes one or more semiconductor integrated circuits. A smart card can interface with a point-of-sale terminal, an ATM, or with a card reader integrated with a computer, telephone, vending machine, or a variety of other devices. The smart card may be programmed with various types of functionality such as a stored-value application, a credit or debit application, a loyalty application, cardholder information, etc. Although a plastic card is currently the medium of choice for smart cards, it is contemplated that a smart card may also be implemented in a smaller form factor, for example, it may attach to a key chain or be as small as a chip module. A smart card may also be implemented as part of a personal digital assistant, telephone (such as a subscriber identification module), or take a different form. The below description provides an example of the possible elements of a smart card, although the present invention is applicable to a wide range of types of smart cards.

A smart card may include a microprocessor, random access memory (RAM), read-only memory (ROM), non-volatile memory, an encryption module (or arithmetic unit), and a card reader (or terminal) interface. Other features may

be present such as optical storage, flash EEPROM, FRAM, a clock, a random number generator, interrupt control, control logic, a charge pump, power connections, and interface contacts that allow the card to communicate with the outside world. Of course, a smart card may be implemented in many ways, and need not necessarily include a microprocessor or other features.

The microprocessor is any suitable central processing unit for executing commands and controlling the device. RAM serves as temporary storage for calculated results and as stack memory. ROM stores the operating system, fixed data, standard routines, look up tables and other permanent information. Non-volatile memory (such as EPROM or EEPROM) serves to store information that must not be lost when the card is disconnected from a power source, and must also be alterable to accommodate data specific to individual cards or changes possible over the card lifetime. This information includes a card identification number, a personal identification number, authorization levels, cash balances, credit limits, and other information that may need to change over time. An encryption module is an optional hardware module used for performing a variety of encryption algorithms. Of course, encryption may also be performed in software. *Applied Cryptography,* Bruce Schneier, John Wiley & Sons, Inc., 1996 discusses suitable encryption algorithms and is hereby incorporated by reference.

The card reader interface includes the software and hardware necessary for communication with the outside world. A wide variety of interfaces are possible. By way of example, the interface may provide a contact interface, a close-coupled interface, a remote-coupled interface, or a variety of other interfaces. With a contact interface, signals from the integrated circuit are routed to a number of metal contacts on the outside of the card which come in physical contact with similar contacts of a card reader device. A smart card may include a traditional magnetic stripe to provide compatibility with traditional card reader devices and applications, and may also provide a copy of the magnetic stripe information within the integrated circuit itself for compatibility.

Various mechanical and electrical characteristics of a smart card and aspects of its interaction with a card reader device are described in *Smart Card Handbook,* W. Rankl and W. Effing, John Wiley & Sons, Ltd., 1997, and are defined by the following specifications, all of which are incorporated herein by reference: *Visa Integrated Circuit Card Specification,* Visa International Service Association, 1996; *EMV Integrated Circuit Card Specification for Payment Systems, EMV Integrated Circuit Card Terminal Specification for Payment Systems, EMV Integrated Circuit Card Application Specification for Payment Systems,* Visa International, Mastercard, Europay, 1996; and *International Standard; Identification Cards— Integrated Circuit(s) Cards with Contacts, Parts* 1–6, International Organization for Standardization, 1987–1995.

Card Architecture

FIG. **1** illustrates symbolically an environment in which Open Platform architecture **10** provides benefits for smart card holders, issuers, application developers and other entities. Although the present invention works well within architecture **10**, it is also suitable for use in other architectures. Open Platform architecture **10** embodied within a smart card **20** provides card issuers an architecture for managing smart cards. Architecture **10** gives card issuers the power to manage and change the content of their cards while also offering them the flexibility to share control of their cards with other business entities. Preferable, ultimate control rests with the card issuer, but through use of architecture **10** other business entities are allowed to manage their own applications on the card issuers cards as appropriate. An issuer personalizes new cards received from a card supplier and then issues these cards to customers. Personalization may also be performed by the card supplier or by a personalization bureau. An issuer may be any suitable issuing entity such as a bank, financial institution, telecommunications network operator, a service association, a merchant or other organization, or even an agent acting for an issuer.

As shown symbolically, in FIG. **1** a wide variety of systems and devices benefit through use of architecture **10**. A smart card **20** has been previously described and its relationship with architecture **10** will be further explained below. A card acceptance device **22** (also termed card reader or terminal) may contain an application that interacts with architecture **10** within smart card **20**, and can also download an application onto the smart card. A terminal management system **24** manages terminals and their respective applications. An application server **26** provides an application for a smart card or card reader. A personalization system **28** personalizes smart card applications. A card management system **30** manages an issuer's card base and respective applications. A key management system **32** provides support for pre-issuance and post-issuance support for key generation and/or key storage and/or key retrieval. Application development tools **34** are used to develop smart card applications.

FIG. **2** illustrates in further detail Open Platform architecture **10** as it may be implemented upon a smart card (not shown). Run-time environment **102** includes both the actual hardware of the smart card as well as the operating system of the smart card. Architecture **10** may be implemented on top of any card run-time environment. Run-time environment **102** is responsible for providing a hardware independent application programming interface (API) for provider applications as well as a secure storable and executing space for applications, thus ensuring that each application's code and data are able to remain separate and secure from others. Architecture **10** may be used with any of a wide variety of physical smart cards available from a wide variety of manufacturers. Further, architecture **10** may be implemented on top of any suitable smart card operating system, such as JAVA Card and Smart Card for Windows, among others.

Architecture **10** includes a number of components as shown in the sample card configuration of FIG. **2**. Card manager **104** is a software application that represents the card issuer. It manages the run-time environment for applications and controls the overall system and security for the smart card. Card manager **104** provides an Open Platform API **110** for applications to access its services and an external command interface for management of the card by off-card management systems. In one embodiment of the invention, this external command interface is an Application Protocol Data Unit (APDU) interface which is an ISO standard communication messaging protocol between a card acceptance device and its smart card. Further details on the APDU interface are contained in *Open Platform Card Specification Version* 2.0 referenced above. Other external command interfaces such as RPC or RMI may also be suitable. Additionally, card manager **104** loads issuer application **112** and performs related card content management on behalf of the card issuer while also managing the loading, installation and deletion of applications provided by application providers.

Card manager **104** also performs APDU command dispatching and application section. Card manager **104** includes an internal card registry **250** as an information resource for card management. The card registry contains information for managing card, load file and application life cycle states, card blocking, personal identification numbers (PINs), application loading, installation and deletion, and the authorization of memory allocation.

Card manager **104** can also function as an application. It has application characteristics such as an application identifier (AID), application life cycle states, and it can select itself as the selected application. For example, card manager **104** functions as an application when the card issuer selects the card manager to load a new application onto a new card.

Card manager **104** is responsible for overall card security and includes the security domain application of the card issuer which supports key handling, encryption, decryption, signature generation and verification for the card issuer's applications. Card manager **104** may include a variety of cryptographic keys for the smart card to perform these functions. In a preferred embodiment of the invention, it includes at least one set of static symmetric keys including an authentication/encryption key, a MAC creation key, and a key encryption key. The authentication/encryption key is used for the initial mutual authentication of card and host and for data encryption during regular session processing. The MAC creation key is used to calculate data authentication patterns for (a) verifying the integrity of data in a command data field and (b) verifying the authenticity of a command. The key encryption key is used to decrypt keys that are received by the card. Most preferably, a session key may be derived from any of the above keys using unique card data and session data. In addition to the symmetric key set, the card manager may make use of a asymmetric cryptography by also including the public key of an issuer for decrypting information that has been originally been encrypted using the private key of the issuer.

Provider security domains **106** and **108** are special key and security management applications that are used to ensure separation of keys between the card issuer and different application providers. For example, security domain **106** includes keys for provider application **114** that are not revealed to card manager **104**, issuer application **112** or other entities on the card. Security domain **108** contains keys unique to another provider application. In this fashion, a security domain is the on-card representative of an application provider. It provides cryptographic services for all the applications on a card that are owned by a particular application provider. Alternatively, there may be one security domain for each application on a card. A security domain may be established on behalf of an application provider when the provider requires use of keys on the card which should be kept secret from card issuer keys and other provider keys. The issuer's security domain which includes the issuer's secret keys are preferably included within card manager **104**.

Open Platform API **110** provides an interface that provides access to services provided by card manager **104** and security domains **106** and **108** for various applications. Open Platform API **110** may be implemented in any suitable language and in one embodiment uses the JAVA programming language.

Issuer application **112** is an issuer-provided software application that performs any suitable function on the smart card desired by the issuer. In one embodiment, issuer application **112** may perform the functions of home banking or card content auditing, for example.

Provider application **114** is any suitable software application provided for a smart card by a business entity known as a provider. Applications can generally by classified as non-changeable applications that are loaded into ROM during the manufacturing stage and are not altered during the lifetime of the card, and changeable applications that can be loaded, installed or removed during initialization or post-issuance. Some applications may have components that reside in both immutable memory such as ROM and in mutable memory such as EEPROM. The present invention is suitable for use for any of a wide variety of types of provider applications. These types of applications include: loyalty applications, stored value applications, credit/debit applications, transit, health care, insurance, electronic ticketing, electronic hotel check-in, and coupon applications.

FIG. **3** is another illustration of the Open Platform architecture **10** of FIG. **2** suitable for explaining the present invention. As mentioned earlier, smart card operating system **120** is any hardware dependent operating system for a particular smart card. Hardware independent API **122** is in communication with operating system **120** and provides a consistent API for smart card applications **112–116**. API **122** may be implemented using JAVA Card, Smart Card for Windows, and others. In general, APIs may be implemented using interpretative approaches (JAVA, BASIC, FORTH, etc.), compiled high level languages ("C") or native assembly coding.

Applications **112–116** are present on the smart card and communicate directly with API **122** to perform functions on the smart card. In addition, applications **112–116** use Open Platform API **110** to access the unique system services provided by card manager **104** and any available security domains. As mentioned above, these services to the Open Platform architecture include application state tracking, personalization support, card security management, etc. Security domains **106** and **108** also communicate with card manager **104**.

Life Cycles and Card Registry

FIG. **4** illustrates the card manager life cycle state transitions **200** according to one embodiment of the invention. This life cycle is illustrative of possible states and transitions that work well with the present invention. Other life cycle states for the card manager and different transitions are also possible. As card manager **104** performs a supervisory role over the entire smart card, its life cycle can be thought of as being similar to the life cycle of the smart card. Card manager **104** owns and maintains the life cycle state information and manages requested state transitions in response to external commands (such as APDU commands).

Pre-production state **202** refers to all smart card activities prior to the initial life cycle state of card manager **104**. These activities include mounting a chip onto the smart card, loading an operating system onto the card, etc., and are specific to the manufacturer of the card. A wide variety of other activities may occur during pre-production state **202** depending upon the card manufacturer, the type of card and the issuer.

In general, Open Platform (OP) Ready state **204** and Initialized state **206** are intended for use during the pre-issuance phase of the smart card. States Secured **208**, Locked **210**, and Terminated **212** are intended to be used during the post-issuance phase of the smart card.

In Ready state **204** all of the basic functionality of run-time environment **102** is available and card manager

104, acting as the default application, is ready to receive, execute and respond to external APDU commands. In the Ready state any files loaded into ROM are available, runtime environment 102 is ready for execution, card manager 104 acts as the default application, and an initialization key is available within the card manager 104. During this state, security domains and their key sets may be loaded, applications from ROM may be installed, and applications may be loaded into EEPROM.

The Initialized State 206 is an administrative card production state. Its definition is manufacturer and/or implementation dependent and may have a wide variety of definitions.

The state Secured 208 is the normal operating state for the smart card after issuance. This state indicates that card manager 104 should enforce the issuer's security policies for post-issuance behavior such as application loading and activation. These security policies may vary by card depending upon the specific requirements of each card issuer. Preferably, when an issuer determines that a card is ready to be issued to a cardholder, the issuer irreversibly sets the state of the card to Secured. In one embodiment, the card has the following functionality in this state: the card manager contains its necessary key sets and security elements; issuer initiated card content changes can be carried out through the card manager; post-issuance personalization of applications belonging to the issuer can also be carried out by the card manager; security domains contain their necessary key sets and security elements; provider initiated card content changes can be carried out by security domains that have the delegated management privilege; and post-issuance personalization of applications belonging to a provider can be carried out via a security domain.

Card Manager (CM) Locked state 210 is used to tell card manager 104 to temporarily disable all applications on the card except for the card manager. This state provides the issuer with the ability to detect security threats either internal or external to the card and to be able to temporarily disable the functionality of the card. While in this state, the card will no longer function except via card manager 104 which is controlled by the issuer. While in this state, there is an option to determine that the threat is either no longer present or is of limited severity such that the issuer can reset the card to the normal operating state of Secured.

Card manager 104 is set to the state Terminated 212 to permanently disable all card functionality including card manager 104. This state allows the issuer to logically destroy the card if there is a severe security threat or if the card has expired. Preferably, the Terminated state is irreversible and indicates the end of the smart card life cycle.

FIG. 5 illustrates application life cycle state transitions 220 according to one embodiment of the invention. The application life cycle begins when an application is installed on a smart card. Installation may occur directly during loading or via another file present on the card. Card manager 104 is responsible for managing the initial life cycle state transition of an application before it is fully functional. Once an application is available for selection from the outside world, it takes control of managing its own life cycle. These life cycle states are used to inform card manager 104 of the status of the application. The definition of these states are application dependent and are preferably known only to the application. Card manager 104 can take control of the application life cycle if the card or the issuer detects a security problem or if the application is to be deleted.

The Installed State 222 means the application executable has been properly linked and any necessary memory allocation has taken place so that the application may execute. The installation process does not include establishing the application as an externally visible application (the Selectable State), also, installation is not intended to personalize the application. Preferably, card manager 104 sets the life cycle of an application to the state Installed during the application installation process.

The state Selectable 224 is used to make an application available to receive external commands (such as APDU commands) from outside the card. Card manager 104 is responsible for making an application available for selection by setting its life cycle state to Selectable. Preferably, applications are properly installed and functional before they can be set to the state of selectable.

Transition to the Personalized state 226 is application dependent: preferably this state indicates that the application has all of the necessary personalization data and keys for full run-time functionality. The behavior of the application while in this state is determined by the application itself, and preferably card manager 104 is not involved. Also, the application manages its life cycle transition from the state Selectable to the state Personalized.

The definition of what is required for an application to transition to the state Blocked 228 is application dependent. Preferably, a transition to this state indicates that an application specific security problem has been detected either from within the application or from outside the card. Preferably, the behavior of the application while in this state is determined by the application itself and card manager 104 need not be involved. At any point in the application life cycle, card manager 104 may take control for security reasons and set the application life cycle state to Locked 230. Card manager 104 or an issuer uses the state Locked as a security management control to prevent the selection and execution of an application. The card manager may set an application to Locked if it detects a threat from within the card that appears to be associated with a particular application. Alternatively, an issuer may determine that a particular application needs to be locked for a business or security reason and initiate the transition to the Locked State via the card manager. Once an application is set to Locked, only the card manager may transition the application back to the state that it held just prior to being placed in the Locked State.

If an application is to be removed from the card either logically or physically, the card manager manages that process and then sets the life cycle state to Deleted 232. The request to delete an application may come from the application itself, its associated security domain, the card manager or an issuer. If the card manager receives a request to delete an application which cannot be physically deleted (e.g., because it is stored in ROM), the application may be logically deleted by setting its state to Logically Deleted. Once an application is in the state Logically Deleted, it cannot be reversed. The card manager considers this state to be the equivalent of physical deletion of the application.

Preferably, a security domain also follows the above application life cycle. A security domain may have specific run-time behavior that is different from a regular application. Security domains are installed on the card by the card manager, which sets their state to Installed. A security domain is not available for selection while in this state. The card manager sets a security domain to the state Selectable in order to enable personalization. Once a security domain has been personalized with its keys and other necessary security elements, it sets its state to Personalized. A security domain has the option to block itself as a protection mecha-

nism against a threat. Preferably, blocking a security domain does not have a required effect on access to that security domain by an application via Open Platform API **110**. Security domains do, however, have the option to implement their own specific behavior while in the Blocked State.

As with any other application, the card manager may set a security domain to the state Locked and that domain may no longer be available for selection. Locking a security domain, however, does not have any effect on the access of that domain by applications via Open Platform API **110**. A Locked security domain preferably is only unlocked via a command from the card manager.

FIG. **6** illustrates a card registry database **250** according to one embodiment of the invention. Card registry **250** is persistent data storage on a smart card that supports various functions of card manager **104** and may be implemented using any suitable medium and protocol. In one embodiment card registry **250** supports the functions of command dispatch, card content management, security management and the issuer's security domain in the card manager.

Preferably, registry **250** includes one set of card manager data elements and a separate set of application data elements for each application on the card including security domains. Card manager AID **252** is a unique identifier for the card manager and is used in a Select command intended for the card manager. Card manager life cycle state **254** contains the current life cycle state of the card manager. Other data elements **256** may also be present.

Application identifier **260** is a unique identifier for each application on the card and is used by the card manager for application selection. Application life cycle state **262** contains the current life cycle state of the application or security domain. Resource allocation **264** is a data element that contains a value for the total amount of resources that are available to an application. It is an application specific value and is used as a control mechanism by the card manager to limit the amount of resources that an application uses during run time. When additional resources are requested by an application. The card manager compares against this data element. Application privileges **266** are a set of data elements that indicate privileges for each application. A variety of privileges may be indicated for an application: the application is a security domain without delegated management privilege; the application is a security domain with data authentication pattern privileges; the application is a security domain with delegated management privilege; the application has card manager locking privilege; the application has card termination privilege; the application is the default selected application; and the application has privilege to change a card global PIN. Each privilege may be marked as true or false, and an application may have more then one privilege marked as true. The card manager may apply a set of rules to these privileges for management of the card in any suitable fashion.

Security Domain AID **268** is a data element that contains the AID of an application's corresponding security domain. When an application provider requests a connection to its security domain, the card manager uses this data element to return a reference to this appropriate domain. The card manager acts as the security domain for card issuer applications if no other applicable security domain is present on the card. Other data elements **270** may also be present.

Delegated Loading and Installation of an application

The present invention provides a technique to extend the functionality of a security domain and to allow it to perform delegated management of smart card applications. For example, the present invention allows a security domain to perform delegated loading, installation and/or deletion of an application. By delegating this management to their security domain, a provider of an application is assured of more direct control and management of their application, yet an issuer still maintains some control over the management of the card.

The Open Platform architecture allows parties other than the card issuer such as application providers to load, install, and delete their own applications. In general, these processes are referred to as delegated management. In general it is desirable that a card issuer have complete control over the smart cards it issues. The card issuer, however, may not necessarily wish to manage all card content changes, especially when the content does not belong to the card issuer but to an application provider. This concept of delegated management is incorporated into the Open Platform architecture to allow the card issuer the option of empowering application providers to initiate changes to the issuer's smart cards that are pre-approved by the card issuer.

This pre-approval ensures that only card content changes that the card issuer has approved will be accepted and processed by card manager **104** of a smart card. This delegation of control in the card update process allows application providers more flexibility in managing their own applications on the card issuer's cards.

Delegated loading allows the application provider to establish a loading session for transferring their application files directly to their own security domains. Once each APDU command has been securely transferred onto the card, the security domain passes it to the card manager for loading into persistent memory. The card manager is able to identify the authenticity of these processes through the use of a data authentication pattern applied to the install commands and the load file itself. The card manager does not verify the data authentication patterns applied to individual Load commands. The command related DAPs which the card manager does check are referred to as the Load and Install tokens.

In addition to these DAPs which are intended for the card manager, the application provider applies DAPs (which cover the command and tickets) for securing (integrity) the transport of all commands (Install and Load) to a security domain. Using this terminology, a token is a DAP intended for the card manager, which is distinct from the DAPs that are applied and intended for a security domain.

The card issuer pre-authorizes the initial install command (which performs loading) and the load file through the use of these data authentication patterns. The data authentication pattern for the application file is included in the initial Install command to ensure that application which has been approved by the card issuer is the same application that is subsequently received by the card manager through the series of loading commands that follow the first install command.

A delegated installation process is used to install an application that is already present in the card. The card issuer pre-authorizes a second install command that includes the application installation information. Once again a data authentication pattern is appended to this command to ensure its authenticity. Once the card manager has validated the install command and carried out instructed operations, the card manager may generate a response to return to the security domain. Completion of a delegated load results in the generation of a load receipt while completion of a

delegated installation results in the generation of an install receipt. The processes in the delegated loading and delegated installation may occur in a single transaction where an application is loaded and installed immediately. Alternatively, the delegated loading and installation processes may occur independently of one another. Preferably for delegated loading and installation, the card manager is in the life cycle state Secured, the security domain to be used is in the state Personalized, and the security domain has the delegated management privilege.

A security domain may be loaded onto a smart card and be provided with cryptographic keys in a wide variety of manners. One difficulty with the prior art is that if a security domain or other application were loaded onto a card post-issuance, is that it would rely upon keys on the card that were known to an issuer or another provider. This reliance upon an issuer or other provider for the loading of a security domain might compromise the secret keys within that domain; a provider would wish that those keys were kept separate and secret from an issuer or other provider. Two techniques have been used to provide a security domain on a card having secret keys that do not become known to an issuer or other provider. In one example, any number of security domains with their included secret keys are installed onto the smart card by a manufacturer when the card is produced. These domains are known only to the manufacturer. When the card is released to an issuer, the manufacturer may release certain sets of the keys to the issuer while keeping secret key sets pertaining to various other of the security domains. These retained key sets are either held by the manufacturer, put in escrow or given to a trusted third party. Later in the life of a card when a new application provider wishes to take ownership of a security domain on the card and use it to load an application, this provider receives a secret key set for one of the unassigned security domains from the trusted third party. In this way, the key set for a security domain is known only to the new application provider and is kept separate from the issuer or another application provider.

In another example of how a security domain and its secret keys may be assigned to a provider, a technique as described in U.S. patent application Ser. No. 09/046,993 may be used. In this scenario, one of the security domains on the card is assigned to a trusted third party. The trusted third party takes ownership of the domain and control of its secret keys. Once the card is issued and is in use by a cardholder, the trusted third party retains ownership of the security domain. A new application provider that wishes to load an application onto the card would then approach the trusted third party for permission to use their security domain. The application provider would then load their own application (which might be a new security domain) using the security domain and secret keys of the trusted third party. In this fashion, a new application provider is allowed to load a new application and security keys without having to share the same with an issuer or another application provider. These techniques and others may be used to provide a security domain on a smart card having a security key set that is known to the application provider to whom the security domain belongs.

FIG. 7A is a flow diagram describing a technique for performing delegated loading. As mentioned above, there are a variety of ways that a security domain may be present on a smart card and its secret keys provided to an application provider without the issuer or other application providers being involved. Step **302** refers to this process by which a smart card is manufactured, installed with operating software and security domains, and the keys of at least one security domain are kept secret from the issuer.

A group of keys associated with a particular security domain is referred to as a key set and is uniquely identified by a key set identifier. Each key within a key set is also uniquely identified by a key identifier. These keys in a key set may perform a wide variety of functions during smart card operation and include authentication, confidentiality, integrity, and key encryption. Later, the smart card is issued to a cardholder and becomes activated in step **306**. At this point in time the card is in its secured state **208** and the cardholder is using the card.

At some point in time, an application provider may desire to write a new application for the card and place it onto the card post-issuance using delegated loading. A provider would choose this option to ensure that the secret keys of a security domain do not become available to another party. Accordingly, as a first step, a provider in step **310** writes a suitable smart card application in any suitable language such as JAVA, Visual Basic, Assembly Language, "C", etc. In a preferred embodiment, this application conforms to Open Platform API **110** and hardware independent API **122**.

In step **314** one of the security domains on the card is assigned to the application provider. The provider may be assigned a security domain from the manufacturer who has kept a number of security domains in escrow from the time of card manufacture, or the provider may be allowed to load a new security domain in conjunction with a trusted third party. In conjunction with being assigned a security domain, in step **318** the application provider receives the secret key set corresponding to this security domain. These keys may have been held in escrow by the card manufacturer on behalf of the issuer, or these keys may be dynamically created by the application provider and loading onto the card along with a new security domain through the help of a trusted third party. Because the trusted third party has ownership of a security domain and its keys, it can assist the application provider in loading a new security domain with its new keys. Using either technique, the application provider receives ownership of a security domain on the smart card and its secret set of keys that have not been revealed to the issuer or another third party.

In step **322** the issuer approves the application through a suitable arrangement with the application provider; this step is described in further detail in FIG. 7B. Through this step, the smart card onto which the application will be loaded becomes assured that the issuer has checked and approved of the application. It allows the issuer to keep a certain amount of control over the delegated loading process.

In step **326** a cardholder inserts the smart card into any suitable card acceptance device. This insertion may be part of a regular transaction for the cardholder or it may be a special transaction solely for the purpose of downloading the new provider application. In step **330** the provider downloads the new application onto the smart card in the card acceptance device; this step will be described in further detail in FIG. 7C. In this fashion, the loading of the application has been delegated to an application provider.

In step **334** the provider installs the application that has been loaded onto the smart card; this step will be described in further detail in FIG. 7D. Additionally, the provider may perform other functions for the application such as personalization. At this point, delegated loading and installation of an application onto the card has been performed.

FIG. 7B is a flow diagram that describes how an issuer approves an application for delegated loading and installa-

tion. Once an application provider has written an application for a smart card and desires to load that application onto an issuer's smart cards via the delegated loading process of the present invention, it provides the application to the issuer for approval. It should also be noted that the provider may also give the application to a trusted third party for approval. In step **340** the issuer performs testing of the application given to it by the application provider. Testing of an application for a smart card may be performed in any of a variety of ways and is a step understood in the art, and generally involves functional tests (optional) and security tests (mandatory). Testing of the application involves checking its operational behavior on a smart card, checking its operational memory requirements, etc., ensuring that the application is secure, and checking for viruses and card related threats. Once the issuer (or trusted third party) has tested the application and it to ensure that it behaves correctly, the application is "certified" and the issuer is ready to prepare the application for a delegated load and installation by the provider.

In step **342** creation of a data authentication pattern (DAP) may be performed in a variety of ways. In general a data authentication pattern is a sequence of bytes unique to a string of data such as a command or an application. By calculating a DAP for an application (for example) and delivering the DAP with the application, an entity such as a smart card can recalculate the DAP using the same cryptographic technique. By next comparing the received DAP to the newly calculated DAP the smart card can verify the integrity of the received command or application.

Creation of a DAP may be done in conjunction with either symmetric or asymmetric cryptography, or other suitable technique. In conjunction with symmetric cryptography, the information to be verified (such as a load command, and install command, an application, or a load file) is first assembled. Next, a function such as SHA-1 or MD5 is applied to the information to produce a hash (a unique sequence). Next, a symmetric key is applied to the hash to encrypt it. This unique encrypted sequence is referred to as the data authentication pattern (DAP). In one embodiment, a message authentication code (MAC) may be used to implement a DAP. The DAP may then be appended to the clear text of the information for transmission. On the receiving end (for example inside the smart card), the application (for example) and its appended DAP are received. Next, the same technique is applied to the application using the same cryptographic algorithm as before to produce a new DAP for the application. Assuming the application has not been changed enroute, the newly created DAP should match the DAP received appended to the application. A difference in the two DAPs will indicate that the integrity of the application has been compromised.

In a preferred embodiment of the invention, asymmetric cryptography is used to produce a data authentication pattern for either of the commands or for the application itself. For example, public key/private key cryptography may be used to provide a unique data authentication pattern that the smart card may verify. In this example, the issuer maintains a public key/private key pair. The private key is used by the issuer when approving the application; i.e., the issuer creates data authentication patterns for the commands and for the application. The private key is used to sign a cryptographic hash of the command or the application which then becomes the unique data authentication pattern. The issuer's public key held by the card manager is used to verify the data authentication pattern received along with either a command or an application. That is, the smart card uses the issuer's public key to verify the DAP for the application. As above, if the application has not been changed, the newly created DAP should match the received DAP.

Next, in step **344** the issuer determines general characteristics of the application to send along with the application code in a secure manner. These general characteristics of an application include the name of the application (or its applications identifier), an identifier for the security domain to which the application has been assigned by the issuer, the memory limitations of the application, and any privileges that application may require on the card.

The memory requirements of an application include how much persistent memory an application requires for storage and how much RAM it requires during execution. Privileges of an application include whether or not the application can block the card, whether the application can lock the card manager, whether the application can change the card global PIN, and others. If the application to be loaded in a delegated fashion is a security domain, the privileges granted to the security domain may include whether or not the security domain has the delegated management privilege and whether or not the security domain has the capability to verify data authentication patterns from other applications loaded in a delegated fashion.

In step **346** a command is assembled for loading the application onto the smart card. This load command includes the application characteristics determined in step **344** and may include other information particular to the type of command protocol being used. Preferably, this load command has appended to it the data authentication pattern created for the application in step **342**. By combining the pattern and the application, it is ensured that the application received and approved by the issuer is the same file that will subsequently be received by the card manager on the card. An exemplary load command is shown in FIG. **8**.

In step **348** a data authentication pattern is created for the load command generated in step **346**. It will be appreciated that generation of this pattern for the entire load command may be performed in many ways. In a preferred embodiment of the invention, the pattern for the load command is calculated for the entire load command including the general characteristics it contains as well as the data authentication pattern for the application. Preferably, a load command calculation key known only to the issuer is used to calculate this data authentication pattern of step **348**. In one embodiment of the invention, this load command may appear as shown in FIG. **8** and is based upon a standard APDU command under ISO 7816-4.

Step **350** determines the installation behavior of the application. The information determined in the step provides instructions for the card on what to do with an application when it is installed. This information allows the card to correctly install an application on the card that has previously been loaded onto the card. For example, this information includes the life cycle state in which the application should be when installed on the card, instructions for installing the application, and other information.

In step **352** a command is created for installation of the application on the card and includes the information determined in step **350**. Preferably, this installation command does not include the data authentication pattern for the application, although it may. In step **354**, a data authentication pattern of the entire install command created in step **352** is appended to the end of the install command. Preferably, an install command calculation key known only to the issuer is used to create this data authentication pattern. In one embodiment of the invention, this install command with its

appended data authentication pattern may appear as shown in FIG. 9 and may be implemented using the APDU Install command under the ISO 7816-4 protocol.

In step **356** the assembled load and install commands (along with their data authentication patterns) and the tested and certified application (along with its data authentication pattern) are delivered to the application provider. The application provider is now enabled to perform either the delegated load or a delegated installation. Through the use of the data authentication pattern, not only are the application and the commands locked (e.g., a change in one of them would be detected), but the identity of the issuer and its approval of the application is supplied via the secret issuer keys used to calculate the data authentication pattern for the application and the load and install commands. In this fashion, even though a party other then the issuer is allowed to load or install an application, the smart card will recognize that it is authorized to load or install the application by verifying the data authentication patterns that have been created. Such use will now be described.

FIG. 7C is a flow diagram describing one embodiment of the download application step of FIG. 7A. In step **360** a data link is established between host computer **602** and smart card **604** while the card is in a card acceptance device. Preferably, as part of this procedure, a mutual authentication process is performed between the card and the host using a key set provided to the application provider by the issuer or other trusted third party (as previously described). Preferably, the security domain keys provided to the application provider are used so as to assure security domain **106** that the incoming information is coming from an authorized source are only seen by the application provider. In step **362**, host **602** sends load command **500** to security domain **106** using APDU interface **610**. In essence, this is a request for a load of an application.

Once the security domain has received the load command, in step **364** it passes the command to card manager **104**. Because this command is coming from a security domain, the card manager knows that the load command is part of a delegated load process.

In step **366** the card manager authenticates the load command. Preferably, the card manager uses the same cryptographic technique used by the issuer to create load command DAP **514** to verify the load command DAP on its own. The card manager then compares its created pattern with pattern **514** included with the load command in order to authenticate that the load command received from the application provider is the same load command that had been previously created and approved by the issuer. As described above, creation of load command DAP **514** may be done using a variety of cryptographic techniques and using either asymmetric or symmetric cryptography.

In step **368** the card manager passes the authenticated load command to install routine **614** which process the load command. Preferably, processing of the load command is performed as is normally done for a first APDU Install command that has been received from an issuer. In other words, the installer is unaware that the load command is part of a delegated load operation and process the command as if it were loading an application received directly from the issuer. In general, this processing checks to see if an installation of the application can be allowed. For example, the process checks whether the application is already installed, whether there is memory available, whether the needed libraries are present, whether the right version of the operating system or runtime environment is present, etc. Once approved by the installer, the installer sends an approval message back through the card manager and security domain to host computer **602**. Upon receiving the approval message, the host computer is alerted that loading of the application has been approved.

Accordingly, in step **374** the host sends the application to security domain **106** over links **620**. Preferably, included with the application is the data authentication pattern previously created for it by the issuer. In a preferred embodiment, the application is embedded within a load file such as is illustrated in FIG. **10** which itself is part of an APDU Load command. Of course, the application may be embodied in other types of commands, need not necessarily be part of a load file, or simply may be transmitted by itself along with it data authentication pattern. As with the load command, in step **376** the security domain passes the application on to the card manager. In step **378** the card manager authenticates the application by verifying its data authentication pattern that was created by the issuer previously. As mentioned above, the card manager may authenticate the data authentication pattern of the application using any of a variety of cryptographic techniques. If any authentication fails, the original memory contents are restored.

In step **382** the installer loads the actual application code into memory of the smart card and performs linking to any run-time libraries and other necessary steps. As mentioned above, in a preferred embodiment, the installer performs a load by processing one or more APDU Load commands that contain the application. Assuming that loading and linking was performed successfully, in step **384** a confirmation message is sent from the installer to provider host computer **602** via the card manager and the security domain. Once host **602** has received the confirmation, it is notified that the application has been loaded successfully.

In one embodiment of step **384**, the confirmation message takes the form of a load receipt. The load receipt provides confirmation from the card that a successful load of the application has occurred through the delegated loading process. Preferable, the load receipt includes unique data related to the delegated loading transaction and a data authentication pattern applied by the card manager. By having the card manager apply the data authentication pattern using a key known only to the issuer, the issuer can be assured upon later receipt of the load receipt that in fact the delegated load of the application was performed successfully. In one embodiment, the load receipt is returned in the data field of the response message from the last APDU load command sent to the security domain.

Construction of a load receipt and calculation of a data authentication pattern may be performed in a variety of ways. By way of example, the data authentication pattern is calculated using data unique to the loading transaction and a card manager load receipt calculation key known only to the issuer. Preferably, the card manager calculates the data authentication pattern and constructs the load receipt. Information upon which the data authentication pattern is calculated using the key may include: a confirmation counter (which is used to indicate the number of times an application has been loaded on a single card), card unique data (such as a unique card identifier), the load file AID, and the security domain AID. The load receipt key is then applied to this information to generate the load receipt data authentication pattern. The load receipt is then constructed by concatenating the load receipt DAP with the confirmation counter and identification data for the card. In this fashion a provider may later provide the load receipt to the issuer to confirm that the provider's application was successfully loaded onto a particular smart card.

FIG. 7D is a flow diagram describing the install application step of FIG. 7A. In step **386**, host **602** sends install command **520** to security domain **106** using APDU interface **610**. In essence, this is a request for an install of an application. Preferably, the security domain keys provided to the application provider previously by the issuer are used so as to assure security domain **106** that the incoming information is coming from an authorized source.

Once the security domain has received the install command, in step **388** it passes the command to card manager **104**. Because this command is coming from a security domain and not from an issuer off-card, the card manager knows that the install command is part of a delegated install process.

In step **390** the card manager authenticates the install command. Preferably, the card manager uses the same cryptographic technique used by the application provider to create install command DAP **534** to recreate the install command DAP on its own. The card manager then verifies the signature pattern included with the install command in order to authenticate that the install command received from the application provider is the same install command that had been previously created and approved by the issuer. As described above, creation of install command DAP **534** may be done using a variety of cryptographic techniques and using either asymmetric or symmetric cryptography.

In step **392** the card manager passes the authenticated install command to the install routine **614** which processes the install command. In step **394** the installer process the install command which instructs the card to take certain actions with respect to the application recently loaded. These actions may include but are not limited to allocating data space, placing the application into an initial state and setting additional privileges such as default selection and making the application available for selection.

Preferably, processing of the install command is performed as is normally done for a second APDU Install command that has been received from an issuer. In other words, the installer is unaware that the install command is part of a delegated install operation and process the command as if it were installing an application at the direction of the issuer.

In step **396** a confirmation message is sent back to the provider host via the card manager and security domain in much the same way as in step **384**.

In one embodiment of the invention, the confirmation message of step **396** takes the form of an install receipt that may be produced in the same fashion as the load receipt of step **384**. The install receipt provides confirmation from the card that a successful installation of the application has occurred through the delegated installation process. Preferable, the install receipt includes unique data related to the delegated installing transaction and a data authentication pattern applied by the card manager. By having the card manager apply the data authentication pattern using a key known only to the issuer, the issuer can be assured upon later receipt of the install receipt that in fact the delegated install of the application was performed successfully. In one embodiment, the install receipt is returned in the data field of the response message from the last APDU Install command sent to the security domain.

Construction of an install receipt and calculation of a data authentication pattern may be performed in a variety of ways. By way of example, the data authentication pattern is calculated using data unique to the installing transaction and a card manager install receipt calculation key known only to the issuer. Preferably, the card manager calculates the data authentication pattern and constructs the install receipt. Information upon which the data authentication pattern is calculated using the key may include: a confirmation counter, card unique data (such as a unique card identifier), the load file AID, and the application instance AID. The install receipt key is then applied to this information to generate the install receipt data authentication pattern. The install receipt is then constructed by concatenating the install receipt DAP with the confirmation counter and identification data for the card. In this fashion a provider may later provide the install receipt to the issuer to confirm that the provider's application was successfully installed onto a particular smart card.

FIGS. **8** and **9** illustrates examples of a load and an install command that may be created in steps **346** and **352** of FIG. **7B**. Those of skill in the art will appreciate that these commands may take any of a variety of forms and may have fields that occur in different orders. In a preferred embodiment of the invention, these commands are implemented according to the APDU protocol such as described in the *"Open Platform Card Specification"* referred to above. Install **502** indicates that this is an APDU Install command. This command may be used to either load an application or install an application. A load indicator **504** indicates that a load file containing an application is to be loaded. A load file application identifier **506** contains a unique identifier for the application contained in the load file. A security domain application identifier **508** indicates to which security domain the application to be loaded belongs. Load parameters **510** specify additional information that may be required by the card to load the application. These parameters may include any of the application general characteristics determined in step **344**. A data authentication pattern **512** for the load file uniquely identifies the load file and/or the application and may be created as described in step **342**. The data authentication pattern **514** for the entire load command **500** may be created as described in step **348**.

Install **522** indicates that this is an APDU Install command. An install indicator **524** indicates that an application is to be installed. A load file application identifier **526** contains a unique identifier for the application contained in the load file. An application identifier **528** in the load file identifies a particular application in the load file since one load file may contain multiple applications. An application instance identifier **530** specifies the identifier which will be established for selection of the newly installed application.

Install parameters **532** specify additional information that may be required by the card to install the application. These parameters may include any of the application installation behavior determined in step **350**. The data authentication pattern **534** for the entire install command **520** may be created as described in step **354**.

FIG. **10** illustrates a load file **560** containing an application according to one embodiment of the invention. It should be appreciated that an application to be loaded onto a card may be embedded within a load file, as shown, may be loaded simply by itself, or may be loaded in combination with other applications and information. In a preferred embodiment of the invention, an application is embedded within a load file using tag-length-value format. Load file **560** includes any number of data authentication pattern (DAP) blocks **562** followed by a data block **564** which includes the application. There may be multiple DAP blocks **562** that precede a data block **564**. Each DAP block would correspond to a different entity that wishes to review and certify the application before it is allowed to be loaded onto

a smart card. For example, even though an issuer may wish to provide its own DAP block for an application, there may be a regulatory agency that also wishes to review and certify the application and add its own data authentication pattern for the application. In this way, multiple entities can approve an application to be loaded onto a smart card. During the loading or installation process, the card manager or a security domain which represents an entity can verify that the data authentication pattern earlier added by that entity has not changed. It is also conceivable that there is no DAP block **562** present with an application. In this scenario, a data authentication pattern may still be calculated for the application itself but need not appear in a formal DAP block. The data authentication pattern for the application may then be directly appended to the application itself, or may appear in an install command.

In one embodiment of the invention, an application is embedded within a load file **560** as illustrated in FIG. **10**. In this embodiment, any number of DAP blocks **562** precedes a data block **564** which are each structured in tag-length-value format. A load file is the physical data file that is transferred to a smart card in order to make changes to the card contents. In other words, one or more DAP blocks may be used for the integrity verification of the application.

DAP block **562** includes identification information as well as the related authentication pattern. Tag **566** indicate that what follows is a DAP block. Length **568** indicates the length of the DAP block value to follow. The DAP block value **570** includes identification information and the value of the authentication pattern. An identifier tag indicates that an identifier is to follow. An identifier length indicates the length of the identifier, and an identifier value identifies the entity associated with DAP block **562** that has provided the authentication pattern for the following data block **564**. Next, within value **570** a DAP tag indicates that an authentication pattern is to follow. A DAP length indicates the length of the pattern and finally a DAP value provides the actual data authentication pattern providing integrity verification for the application.

Data block **564** includes a data block tag **572** indicating that a data value (an application) is to follow. Data block length **574** indicates the length of the following value. Data block value **576** includes the application to be loaded. Value **576** may contain one or more applications and/or any combinations of library and support files.

In one specific embodiment of the invention, tag **566** has as value of 'E2', DAP identifier tag and DAP tag of **570** have values of '4F' and 'C3', respectively, and tag **572** has a value of 'C4'. Also, DAP block **562** is optional. Even if present, DAP block length **568** may be set to '00'.

A load file may be transported onto the card either via card manager **104** or a security domain that has the delegated management privilege. Card manager **104** preferable is responsible for the physical memory management and life cycle management of a load file. A load file may have two life cycle states which are "Loaded" and "Logically Deleted." All load files present on the card and available for use are in the Loaded state. Any load file which has been requested to be deleted by the card manager or a security domain but cannot be physically deleted is in the state Logically Deleted and cannot be accessed. Loading an application involves first loading the load file onto the card, and then extracting the application and installing it. Alternatively, the load file is processed dynamically during the loading transaction during which the application is extracted. In this alternative embodiment, the remainder of the load file is disregarded and the load file never exists on the card.

FIG. **11** illustrates an embodiment in which an application may be downloaded from an application provider host computer **602** to a smart card **604** in a delegated manner. Host computer **602** is any suitable computing device under control of an application provider that includes the load command **500**, load file **560** and install command **520** that have been approved and received from the issuer.

Smart card **604** includes card manager **104**, security domain **106** and other typical features of a smart card (not shown). Security domain **106** and card manager **104** each include an APDU interface, **610** and **612** respectfully, that allow an outside entity to communicate with them. Additionally, card manager **104** includes an installer routine **614**. Installer **614** is a known low-level memory management routine that accepts application code and other information and writes it to memory.

In the prior art, an issuer by virtue of its secret keys would be able to talk directly to card manager **104** through APDU interface **612** to provide an application to be loaded onto the card. Installer **614** would accept this application via the card manager and install the application in the memory of the smart card. For security, the keys to access card manager **104** would not be accessible to parties other then the issuer, meaning that only an issuer could download an application. Through use of the present invention, a third party application provider is able to perform a delegated load of an application via security domain **106**. Using keys previously received under an arrangement with an issuer, host computer **602** establishes a secure communication channel **620** to security domain **106** of smart card **104** in any suitable card acceptance device (not shown).

Security domain **106** then manages the downloading of load command **500**, load file **560** and install command **520** onto smart card **604**. In this fashion, these commands and the load file may then be delivered via an internal link **622** to card manager **104** using a delegated management interface **616**. The card manager then passes the commands and load file to installer **614** for loading and installing an application onto a smart card. Installer **614** receives and process these commands from security domain **106** in much the same way as if these command had been received from an issuer via card manager **104**. Further, the data authentication pattern present in the commands and in the application may be checked by the card manager to ensure the authenticity and integrity of the information as established by the issuer. Further details on loading and installation are provided in FIG. **7C**.

Delegated Deletion of an Application

Application providers have the ability to instruct the card manager to delete applications that they own. The card manager will honor these requests without authorization from the card issuer. Therefore, there is no requirement for a delete token for to be included with a Delete command passed from an application provider's security domain to the card manager. Once a Delete command is received by the card manager, the card manager verifies that the application being requested for deletion belongs to the security domain that issued the command. After verifying this information, the card manager carries out the deletion and then returns a response that includes a DAP generated by the card manager. This response including the DAP is referred to as the Delete Receipt. FIG. **12** is a flow diagram describing a technique for performing delegated loading.

In a preferred embodiment of the invention, a card undergoing a deletion is in a particular state. For example, the card

manager life cycle state is Secured, the security domain to be used is in the state Personalized and has the delegated management privilege. Preferably, the APDU sequence used is first a Select command indicating a security domain, then an optional authentication process, followed by a Delete command referencing a particular application identifier (AID).

In one embodiment, the card manager uses the following techniques for application removal. Application removal may involve the removal of application instance as well as the associated Load file. Physical removal may occur in mutable persistent memory while only logical removal is possible in immutable, persistent memory. For applications instances or Load files loaded into mutable persistent memory, the card manager: deletes the identified application instance or Load File from mutable persistent memory; reclaims the mutable persistent memory space for future use; and removes the AID of the removed application or Load file from the card registry. For application instances or Load files loaded into immutable persistent memory, the card manager: deletes related application data space (if any) contained in mutable persistent memory; and changes the life cycle state of the deleted application and/or Load file(s) in the card registry from the current state to Logically Deleted.

Alternative Embodiments

In an alternative embodiment, the delegation mechanism mentioned above may be used to allow the issuer to outsource loading and management functions potentially to a single (or multiple) party who can then represent the issuer to other application providers in the event that an application provider does want the issuer (or his representative) to load the application provider's application. For example, a first service provider is the owner of one security domain on a card and a second service provider is the owner of another security domain on the card. The card issuer has a contract with the first service provider for delegated loading and a contract with the second service provider, but not for loading. The application provider has a contract with the second service provider for application personalization. At the request of the application provider, the first service provider loads the application (using the first security domain) and the second service provider personalizes the application (using the second security domain). The provider is then allowed to use the application on the card.

Computer System Embodiment

FIGS. **13** and **14** illustrate a computer system **900** suitable for implementing embodiments of the present invention, such as any of the computers used in the systems shown in FIG. **1**. FIG. **13** shows one possible physical form of the computer system. Of course, the computer system may have many physical forms ranging from an integrated circuit, a printed circuit board and a small handheld device up to a huge super computer. Computer system **900** includes a monitor **902**, a display **904**, housing **906**, a disk drive **908**, a keyboard **910** and a mouse **912**. Disk **914** is a computer-readable medium used to transfer data to and from computer system **900**.

FIG. **14** is an example of a block diagram for computer system **900**. Attached to system bus **920** are a wide variety of subsystems. Processor(s) **922** (also referred to as central processing units, or CPUs) are coupled to storage devices including memory **924**. Memory **924** includes random access memory (RAM) and read-only memory (ROM). As is well known in the art, ROM acts to transfer data and instructions uni-directionally to the CPU and RAM is used typically to transfer data and instructions in a bi-directional manner. Both of these types of memories may include any suitable of the computer-readable media described below. A fixed disk **926** is also coupled bi-directionally to CPU **922**; it provides additional data storage capacity and may also include any of the computer-readable media described below. Fixed disk **926** may be used to store programs, data and the like and is typically a secondary storage medium (such as a hard disk) that is slower than primary storage. It will be appreciated that the information retained within fixed disk **926**, may, in appropriate cases, be incorporated in standard fashion as virtual memory in memory **924**. Removable disk **914** may take the form of any of the computer-readable media described below.

CPU **922** is also coupled to a variety of input/output devices such as display **904**, keyboard **910**, mouse **912** and speakers **930**. In general, an input/output device may be any of: video displays, track balls, mice, keyboards, microphones, touch-sensitive displays, transducer card readers, magnetic or paper tape readers, tablets, styluses, voice or handwriting recognizers, biometrics readers, or other computers. CPU **922** optionally may be coupled to another computer or telecommunications network using network interface **940**. With such a network interface, it is contemplated that the CPU might receive information from the network, or might output information to the network in the course of performing the above-described method steps. Furthermore, method embodiments of the present invention may execute solely upon CPU **922** or may execute over a network such as the Internet in conjunction with a remote CPU that shares a portion of the processing.

In addition, embodiments of the present invention further relate to computer storage products with a computer-readable medium that have computer code thereon for performing various computer-implemented operations. The media and computer code may be those specially designed and constructed for the purposes of the present invention, or they may be of the kind well known and available to those having skill in the computer software arts. Examples of computer-readable media include, but are not limited to: magnetic media such as hard disks, floppy disks, and magnetic tape; optical media such as CD-ROMs and holographic devices; magneto-optical media such as floptical disks; and hardware devices that are specially configured to store and execute program code, such as application-specific integrated circuits (ASICs), programmable logic devices (PLDs) and ROM and RAM devices. Examples of computer code include machine code, such as produced by a compiler, and files containing higher level code that are executed by a computer using an interpreter.

Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and modifications may be practiced within the scope of the appended claims. Therefore, the described embodiments should be taken as illustrative and not restrictive, and the invention should not be limited to the details given herein but should be defined by the following claims and their full scope of equivalents.

We claim:

1. A method of delegated loading of an application onto a smart card, said method comprising:

assigning a security domain of the smart card to an application provider;

providing a key set to application provider for the security domain assigned to the application provider, wherein the key set is not known to the issuer of said smart card;

approving of said application by an issuer of said smart card, wherein the approving of said application by an issuer of said smart card, comprises:

certifying said application;

creating a data authentication pattern for said application;

creating a command for loading said application;

adding said data authentication pattern to said load command;

creating a command for installing said application;

adding said data authentication pattern to said install command; and

delivering said commands to said application provider;

inserting the smart card into the card acceptance device subsequent to the steps of approving said application, creating said application authentication pattern, and appending said application authentication pattern, and prior to the steps of receiving the load command, and verifying said load command, wherein said delegated loading is performed after issuance of said smart card to a consumer;

receiving a load command from the application provider via a card acceptance device, said load command including an indication of an application to be loaded and an appended command authentication pattern;

verifying said load command using said command authentication pattern;

receiving said application from the application provider via said card acceptance device, said application including an appended application authentication pattern;

verifying said application using said application authentication pattern; and

loading said application into memory of said smart card, whereby said application provider is allowed to load said application onto said smart card.

2. A method as recited in claim 1 wherein said command authentication pattern is generated by an issuer of said smart card using a cryptographic technique, and wherein verifying of said load command includes:

recalculating said command authentication pattern from said load command using said cryptographic technique, whereby said command authentication pattern and said recalculated command authentication pattern may be compared to provide verification of said load command.

3. A method as recited in claim 1 further comprising:

receiving an install command from an application provider via a card acceptance device, said install command including an indication of an application to be installed and an appended install authentication pattern;

verifying said install command using said install authentication pattern; and

installing said application on said smart card, whereby said application provider is allowed to install said application onto said smart card.

4. A method as recited in claim 3 further comprising:

receiving a load command from an application provider via a card acceptance device, said load command including an indication of an application to be loaded and an appended load authentication pattern;

verifying said load command using said load authentication pattern; and

loading said application on said smart card, whereby said application provider is allowed to load said application onto said smart card.

5. A system for delegated loading of an application onto a smart card, said system comprising:

a host computer under control of an application provider;

a software application included in said host computer to be loaded onto a smart card, said application including an appended application authentication pattern produced by an issuer of said smart card that verifies said application to said smart card;

a smart card acceptance device linked to said host computer; and

a smart card included in said card acceptance device, said smart card including code arranged to verify said application using said application authentication pattern, whereby said application provider is allowed to load said application onto said smart card.

6. A system as recited in claim 5 wherein said host computer comprises:

computer code to assign a smart card security domain to the application provider;

computer code for providing a key set to the application provider for the security domain assigned to the application provider, wherein the key is not known to the issuer of the smart card.

7. A system as recited in claim 6 further comprising:

a load command included in said host computer said load command comprising an appended command authentication pattern and code for loading said software application; and

code within said smart card arranged to verify said load command using said command authentication pattern, whereby said application provider provides said load command to said smart card.

8. A system as recited in claim 7 further comprising:

an install command included in said host computer said install command comprising an appended install authentication pattern and code for installing said software application; and

code within said smart card arranged to verify said install command using said install authentication pattern, whereby said application provider provides said install command to said smart card.

9. A system as recited in claim 6 further comprising:

an install command included in said host computer that has an appended install authentication pattern; and

code within said smart card arranged to verify said install command using said install authentication pattern, whereby said application provider is provide said install command to said smart card.

10. A method as recited in claim 6 wherein said cryptographic technique provides authentication and integrity for said application.

11. The system as recited in claim 5 further comprising:

a network connection linked to the smart card issuer;

computer readable code for sending the application from application provider to the smart card issuer;

computer readable code for receiving the approved application and an appended application authentication pattern from the smart card issuer; and

a storage device for storing the application and the appended application authentication pattern.

12. A method of delegated installation of an application on a smart card from an application provider, said method comprising:

sending an application from the application provider to a smart card issuer for approval;

receiving an approval application with an appended install authentication pattern from the smart card issuer;

storing the approved application and appended install authentication pattern at the application provider;

loading the stored approved application onto a smart card from the application provider;

receiving an install command from an application provider via a card acceptance device, said install command including an indication of said application to be installed, install parameters and an appended install authentication pattern;

verifying said install command using said install authentication pattern; and

installing the approved application on said smart card, whereby said application provider is allowed to install said application on said smart card.

13. A method as recited in claim **12** wherein said install authentication pattern is generated by an issuer of said smart card using a cryptographic technique, and wherein verifying of said install command includes:

recalculating said install authentication pattern from said install command using said cryptographic technique, whereby said install authentication pattern and said recalculated install authentication pattern may be compared to provide verification of said install command.

14. A method as recited in claim **12** further comprising:

approving of said install command by an issuer of said smart card;

creating said install authentication pattern;

appending said install authentication pattern to said install command, whereby said smart card is reliably assured that said install command has been approved by said issuer; and

inserting the smart card into the card acceptance device subsequent to recalculating said install authentication pattern creating said install authentication pattern, and appending said install authentication pattern.

15. A method as recited in claim **14** wherein said delegated install is performed after issuance of said smart card to a consumer.

16. A method as recited in claim **15** further comprising:

assigning a security domain of the smart card to the application provider; and

providing a key set to the application provider for the security domain assigned to the application provider, wherein the key set is not known to an issuer of said smart card.

* * * * *

US005856659A

United States Patent [19]

Drupsteen et al.

[11] **Patent Number: 5,856,659**

[45] **Date of Patent: Jan. 5, 1999**

[54] **METHOD OF SECURELY MODIFYING DATA ON A SMART CARD**

[75] Inventors: **Michel Marco Paul Drupsteen**, Alkmaar; **Frank Muller**, Delft, both of Netherlands

[73] Assignee: **Koninklijke PTT Nederland N.V.**, Groningen, Netherlands

[21] Appl. No.: **814,479**

[22] Filed: **Mar. 10, 1997**

[30] **Foreign Application Priority Data**

Mar. 11, 1996 [EP] European Pat. Off. 96200658

[51] **Int. Cl.**[6] **G07F 7/10**

[52] **U.S. Cl.** **235/380**; 235/379

[58] **Field of Search** 235/379, 380

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,161,231 11/1992 Iijima 395/800

FOREIGN PATENT DOCUMENTS

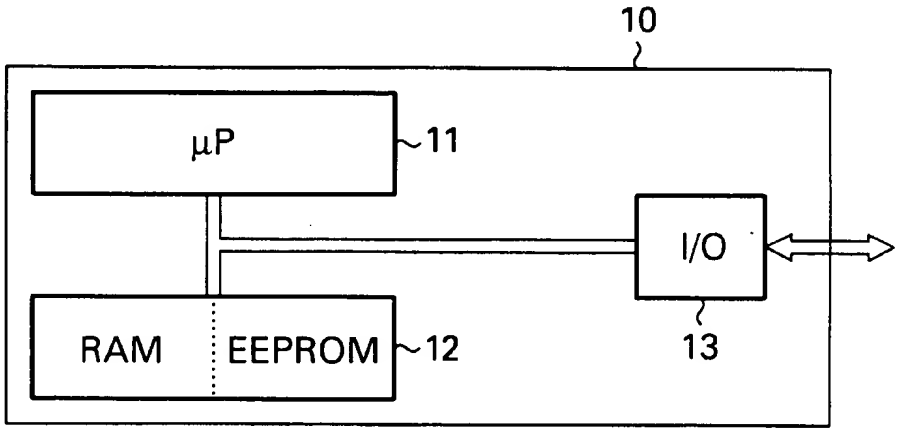
0 218 176 4/1987 European Pat. Off. .
0 484 603 A1 5/1992 European Pat. Off. .
0 559 205 A1 8/1993 European Pat. Off. .
41 19 924 A1 12/1992 Germany .

Primary Examiner—F. L. Evans
Attorney, Agent, or Firm—Oblon, Spivak, McClelland, Maier & Neustadt, P.C.

[57] **ABSTRACT**

A method of securely transferring data, such as program data, to a smart card (1) is disclosed. In order to be applicable in non-secure environments, the method of the invention involves a one-way authentication mechanism, comprising the use of a series of interrelated authentication values (R0, R1, R2, . . .). Further security is achieved by using commands (INIT, TRAN) which are each provided with an authentication code (MAC0, MAC1, . . .). In order to transfer data only to selected cards, the card may accept transfer commands only if a card profile (CP) matches a distribution profile (DP).

12 Claims, 4 Drawing Sheets



COMMANDS
INIT (R0, N, ...), MAC0
TRAN (R1, 1, DATA1, ...), MAC1
TRAN (R2, 2, DATA2, ...), MAC2
....., ...
TRAN (R[N], N, DATA[N], ...), MAC[N]

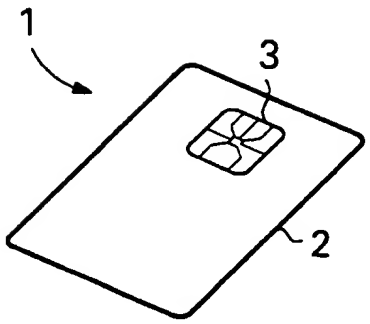


Fig. 1

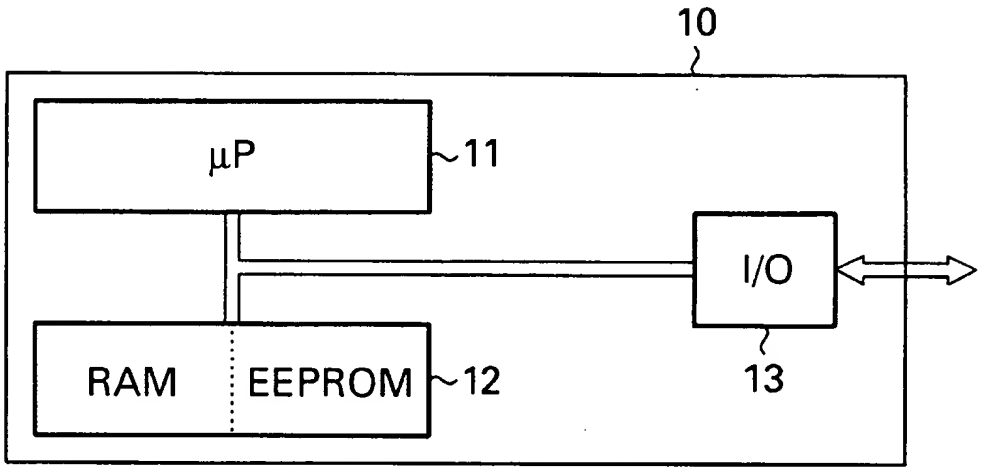


Fig. 2

COMMANDS
INIT (R0, N, ...), MAC0
TRAN (R1, 1, DATA1, ...), MAC1
TRAN (R2, 2, DATA2, ...), MAC2
....., ...
TRAN (R[N], N, DATA[N], ...), MAC[N]

Fig. 3

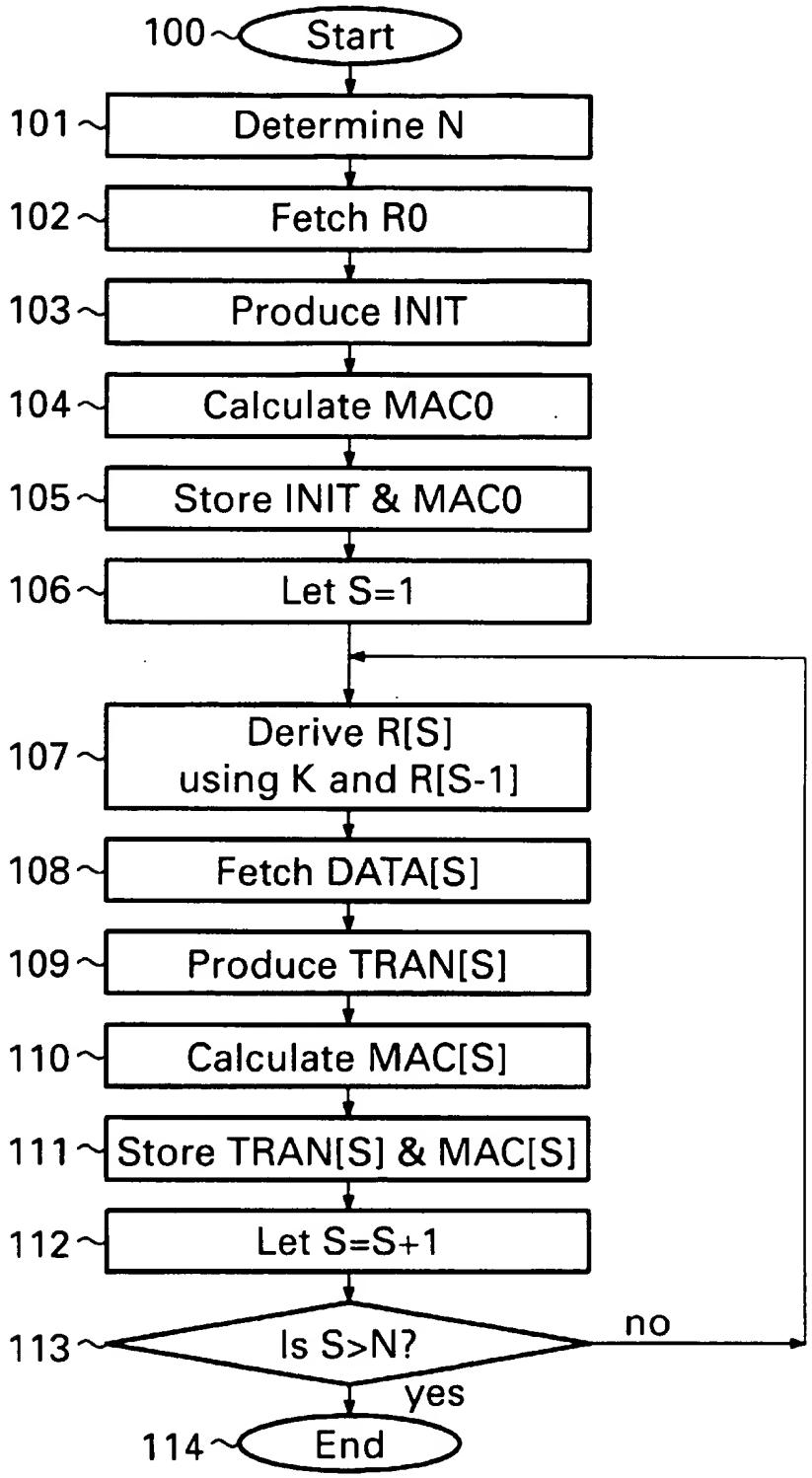

100
Start
101
Determine N
102
Fetch R0
103
Produce INIT
104
Calculate MAC0
105
Store INIT & MAC0
106
Let S=1
107
Derive R[S] using K and R[S-1]
108
Fetch DATA[S]
109
Produce TRAN[S]
110
Calculate MAC[S]
111
Store TRAN[S] & MAC[S]
112
Let S=S+1
113
Is S>N?
no
yes
114
End

Fig. 4

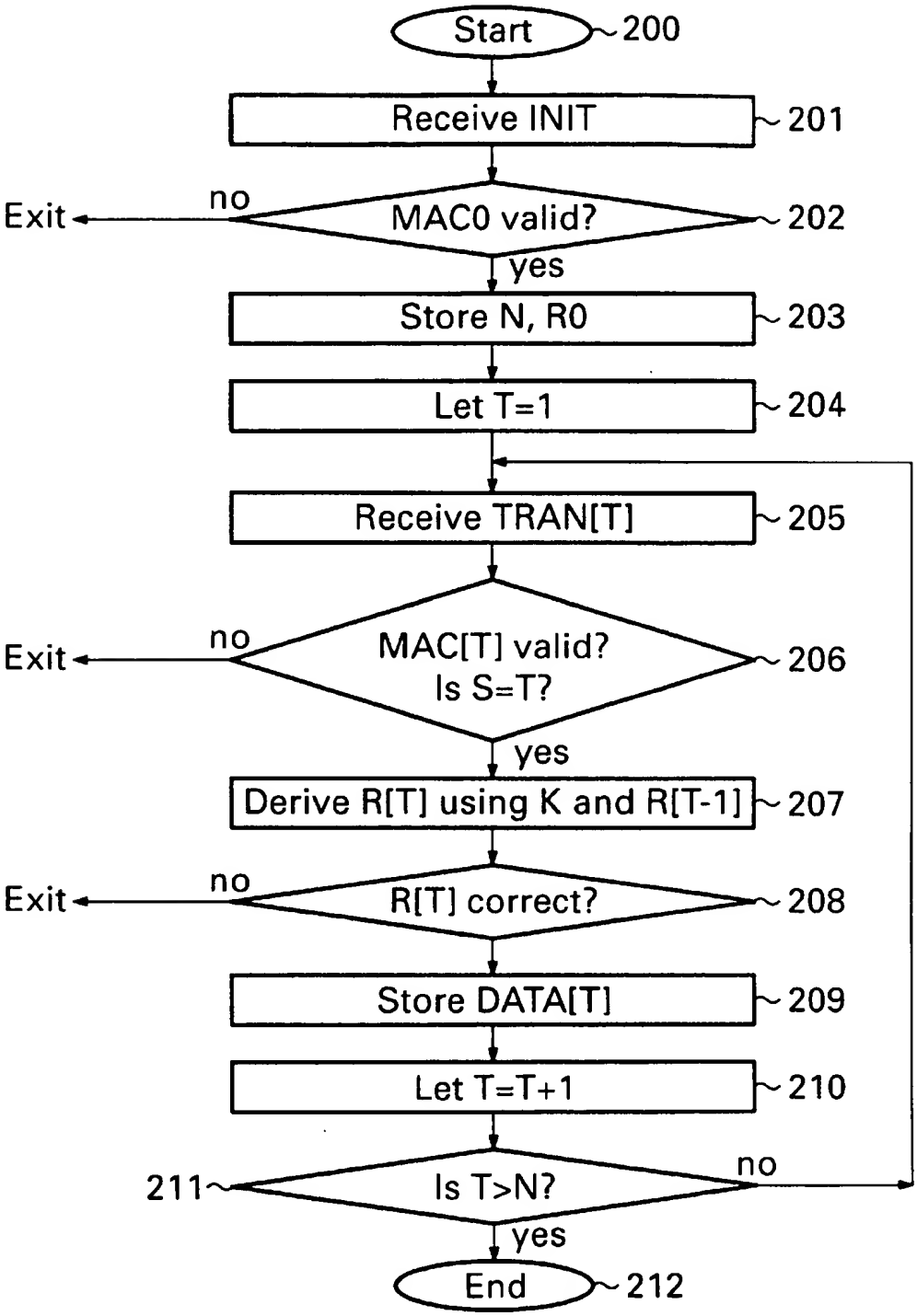

Start
200
Receive INIT
201
Exit
no
MAC0 valid?
202
yes
Store N, R0
203
Let T=1
204
Receive TRAN[T]
205
Exit
no
MAC[T] valid?
Is S=T?
206
yes
Derive R[T] using K and R[T-1]
207
Exit
no
R[T] correct?
208
Store DATA[T]
209
Let T=T+1
210
211
Is T>N?
no
yes
End
212

Fig. 5

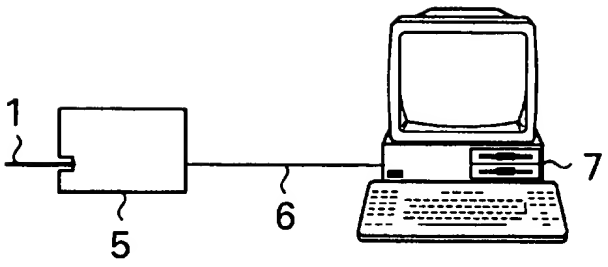


Fig. 6

METHOD OF SECURELY MODIFYING DATA ON A SMART CARD

BACKGROUND OF THE INVENTION

The present invention relates to a method of securely modifying data on a smart card. More specifically, the present invention relates to a method of securely loading or deleting data, and creating and deleting data structures, on a smart card, which method is also applicable in environments where so-called challenge—signed response authentications are not possible. The data concerned may comprise either executable (program) data, i.e. commands, or static (non-program) data, or both. The so-called static data may comprise file structures and/or data structures.

In modern payment systems, the use of electronic payment means becomes increasingly important. Electronic payment means, such as memory cards and smart cards (generally called IC cards), are gaining acceptance as their applications are expanded. In many countries electronic cards are being used for public telephones and the like. Advanced cards are capable of containing electronic "purses", in addition to other functionalities. Such advanced payment means contain, in addition to a memory, a processor capable of running suitable programs.

It should be noted that in this text, the terms smart card or card will be used to denote electronic payment means having at least one integrated electronic circuit comprising a processor and a memory. The actual shape of a so-called smart card is not of importance.

The programs running on the processor of a smart card determine the services offered by the card, that is, the functions and associated data structures (e.g. purse, user identification, loyalty program) of the smart card depend on the software present in the card. As time passes, the need often arises to update the programs of the card, for example in order to add a new function or to improve an existing function. To this end, the card should be able to accept new programs which may replace other programs. However, it must be ascertained that the newly loaded programs are valid. Authentication of programs can relatively easily be accomplished by using a secure data exchange protocol where data are exchanged between a card and a secure terminal (having, for instance, a so-called security module in which keys and other data are securely stored). Such a secure protocol may comprise a challenge-signed response protocol. Examples of protocols for exchanging data between a smart card and a terminal are disclosed in e.g. U.S. Pat. No. 5,161,231 and European Patent Application 0,559,205 (corresponding with U.S. Pat. No. 5,369,760), which are incorporated by reference in this text.

However, in case a secure terminal is not present, such a secure protocol involving e.g. a challenge-signed response cannot be used, as the security of such a protocol depends on the trustworthiness of the terminal.

SUMMARY OF THE INVENTION

It is an object of the invention to overcome the above-mentioned and other disadvantages of the prior art and to provide a method which allows data to be loaded in a smart card in a secure manner, even in non-secure environments. It is a further object of the invention to provide a method for securely transferring data to a smart card, which method comprises a one-way authentication mechanism. It is still a further object of the invention to provide a method for securely transferring data to a smart card, which method comprises a store and forward protocol.

To achieve these and other objectives, a method of securely modifying data in a smart card comprises according to the present invention the steps of producing an initiation command comprising an initial authentication value and producing an initial authentication code based on the command and the initial value, producing at least one transfer command comprising data to be transferred and producing a subsequent authentication code based on both the command and a subsequent authentication value derived from the initial value, transferring the commands and their authentication codes thus produced to the smart card, authenticating the commands in the smart card by checking the authentication codes and checking the subsequent authentication values derived from the initial value, and storing the transferred data in the card.

That is, the initiation command provides the card with an initial authentication value, which value is also used to produce the subsequent authentication values comprised in the transfer commands. By deriving the subsequent values and comparing these derived values with the values comprised in the received transfer commands, the card can effectively check the authenticity of the received commands. In addition, an authentication code, such as a "hash", parity bits or a message authentication code (MAC) in general, is used to verify the authenticity of the received commands. Thus an additional protection is provided which is especially advantageous when the data transferred comprise card commands having no built-in protection mechanism, i.e. for protectedly transferring unprotected commands. The combination of the use of authentication values and authentication codes, as set out above, provides an effective protection against the fraudulent manipulation of the data transferred.

In order to achieve a further check mechanism, the initiation command preferably further comprises the number of transfer commands. This enables the card to check whether all transfer commands have been received, and prevents the unnoticed loss of transferred data.

Advantageously, the initiation command is arranged to prohibit the card from accepting commands other than transfer commands. In this way the transfer of data to the card can not be interrupted by the execution of other commands. Also, it is prevented that partially transferred data are manipulated.

Preferably, each transfer command further comprises a sequence number. This allows not only a check on the uninterrupted receipt, in the correct order, of the transfer commands, but also provides the possibility to retransmit a limited number of transfer commands if transmission errors (as detected by e.g. the authentication codes) have occurred. It further provides the possibility to undo a number of transfer commands.

The transfer commands may in principle be used to transfer any kind of data. The transfer data may thus not only comprise e.g. monetary balances and lists of telephone numbers (static data), but also one or more card commands (executable data). Such commands may e.g. comprise an UPDATE command which updates one or more memory locations, or a CREATE command which allocates memory locations. Apart from existing commands, new card commands may also be comprised in the transferred data. A card command may be stored in memory, but may also be directly executed upon transfer to the card. In this way an efficient way of modifying the memory contents of the card is achieved.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 schematically shows a smart card as may be used in the method of the present invention.

FIG. 2 schematically shows the integrated circuit of the smart card of FIG. 1.

FIG. 3 schematically shows the commands as may be used in the method of the present invention.

FIG. 4 schematically shows by way of example how a set of commands may be produced according to the method of the present invention.

FIG. 5 schematically shows by way of exemple how a set of commands may be processed by the card according to the method of the present invention.

FIG. 6 schematically shows a system for modifying data on a smart card.

EXEMPLARY EMBODIMENTS

The smart card or IC card **1** shown schematically and by way of example in FIG. **1** comprises a substrate **2**, in which an integrated circuit is embedded. The integrated circuit is provided with contacts **3** for contacting a card reader or the like. It should be noted that the present invention can also be applied in the case of so-called contactless smart cards.

The integrated circuit **10** shown schematically and by way of example in FIG. **2** comprises a processor **11**, a memory **12** and an input/output circuit **13**. The memory may comprise a volatile (RAM) memory part for temporarily storing data and a non-volatile (ROM) memory part for permanently or semi-permanently storing data. The latter part is preferably an EEPROM type memory. The data stored in the non-volatile part may contain both programming data (instructions, programs) and payment data, i.e. data relating to monetary transactions. It will be understood that a separate memory (not shown) may be provided to store the instructions of the processor **11**.

The method of the invention, as depicted schematically and by way of example in FIGS. **4** and **5**, involves the transfer of data from an outside source (e.g. an application provider) to a card, such as the card **1** of FIG. **1**. The transfer of data according to the invention involves producing a set of commands, such as depicted in FIG. **3**. An initiation command (INIT) is followed by one or more transfer commands (TRAN). All commands comprise parameters (R0, R1, . . . , N, . . .) and all transfer commands comprise data (DATA1, DATA2, . . .). The parameters comprise authentication values (R0, R1, . . .). Associated with each command is an authentication code (MAC0, MAC1, . . .). The way this set of commands ("script") is produced is explained with reference to FIG. **4**.

The procedure is initiated in step **100**. In step **101**, the number of transfer commands (TRAN) is determined. Data items to be loaded in the smart card are preferably arranged in a table beforehand. The number of data items (N) determines the number of commands (TRAN) necessary to load the data items into the card. The number (N) of transfer commands can then be determined by e.g. counting the number of items in the table (not shown).

In step **102**, the initial authentication value R0 is fetched. The value R0 may be predetermined, but is preferably generated by e.g. a random number generator. The value R0 is temporarily stored for later use.

In step **103**, the initiation command (INIT) is produced by assembling the command code proper, the value R0 and the number N. An authentication code MAC0 is calculated in step **104**. The code MAC0 may e.g. be calculated according to the ANSI (American National Standardization Institute) X9.19 standard using triple encryption. In step **104** the initiation command INIT and its associated authentication value MAC0 are (temporarily) stored.

In step **106**, a loop is initiated in which the transfer commands and their authentication codes and authentication values are determined. The transfer commands are preferably provided with a sequence number (S), which is set to 1 in step **106**.

In step **107**, the authentication values R1, R2, . . . may be derived from the initial authentication value R0 by e.g. using a random number generator. Alternatively, a message authentication code process (e.g. according to the ANSI X9.19 standard) may be employed for producing the authentication values, the resulting "MAC" being used as authentication value. In this process, one or more keys may be used, such as the secret key K.

In the procedure of FIG. **4**, the derivation of authentication values (R) should be performed at least N times. It should be noted that an even greater security can be achieved by using e.g. every other derived value, that is by skipping values, in which case the derivation should take place more than N times. The authentication values R1, R2, . . . should be related, i.e. derived from each other, but need not be successive results of the derivation process. Thus, only e.g. R0, R2, R4, . . . could be used.

In step **108**, the data-to-be-transferred are fetched, e.g. from the data table mentioned earlier. These data are used in step **109** to produce the transfer (TRAN) command, which comprises the command proper (instruction code), an authentication value (R[N]), the sequence number (S), and the data (DATA[N]). In step **110**, an authentication value (MAC[N]) is calculated like in step **104**. The transfer command and its associated authentication code are stored in step **111**.

After incrementing the sequence number S in step **112**, the sequence number S is compared with the number N of data items in step **113**. If all data items have been processed, this part of the method is terminated in step **114**. Otherwise, control returns to step **107**.

In FIG. **5** it is shown how the set of commands produced according to FIG. **4** is processed by the smart card. After activation of the card in step **200**, the initiation command (INIT) is received in step **201**. First, its authentication code (MAC0) is verified in step **202**. This preferably involves the re-calculation of the authentication code and comparing the re-calculated code (MAC0') with the received code (MAC0). If the code is valid, the procedure is continued with step **203**. Otherwise, the procedure is exited. After an exit, special measures may be taken, such as producing a request for retransmission. For the sake of clarity, this is not shown in FIG. **5**.

In step **203**, the values of N and R0 (initial authentication value) contained in the initiation command are stored for later use. Preferably, the card is put into a mode in which only transfer commands can be accepted, thus preventing the interruption of the procedure. Preferably, the card remains in this "locked", that is transfer-only state until all transfer commands have been received. As stated above, the total number (N) of transfer commands is contained in the INIT command.

Step **204** prepares for a loop in which the transfer commands are processed. A counter T, which corresponds with the sequence number S of FIG. **4**, is set to one.

In step **205** a transfer command (TRAN) is received. The first time step **205** is executed, the first transfer command TRAN1 will be received. It will be understood that all commands may be "received" virtually simultaneously, and that in step **205** the actual processing of the individual transfer commands begins. The first transfer command, as shown in FIG. **3**, has the structure:

TRAN(R1, 1, DATA1, . . .), MAC1, where TRAN represents the command proper, R1 is the first subsequent (that is, second) authentication value, 1 is the sequence number (S), DATA1 is the data contained in the command, and MAC1 is the authentication code of the command.

In step **206** of FIG. **5** a double test is performed. First, the authentication code MAC[T] of the transfer command is checked for its validity, e.g. by reproducing the code and comparing the result (e.g. MAC1') with the received code (e.g. MAC1). If the codes are not equal, the routine is exited, following which the command is rejected and a re-transmission request may be issued. The card may contain the key or keys necessary for producing an authentication code. It will be understood that instead of reproducing the code, some other validity check may be employed.

Furthermore, the card checks the received sequence number S to see that all transfer commands are received in the correct order. If the received sequence number S is not equal to the counter T, the routine is exited, as described above.

If both the authentication code and the sequence number are found to be valid, the routine continues with step **207**, in which the authentication value R is derived from the previous value. In case the first transfer command is being processed (T=1), R1 is derived from R0 using a key K. This derivation in step **207** corresponds to the derivation in step **107** of FIG. **4**.

The correctness of R is checked in step **208**. If the derived authentication value (R') is not identical to the received value (R), the routine is exited. This ensures that all transfer commands are authentic, i.e. provided with interrelated authentication values as derived in the routine of FIG. **4**.

If the authentication value R is found to be correct, data contained in the transfer command are stored. In general, these data may be stored in the memory of the card. However, if the data comprise card commands, i.e. instructions for the processor of the smart card, these card commands may be directly executed. In the latter case, the transfer command may load the smart card command concerned directly into the instruction register of the smart card processor and make the processor execute the command. Transfer commands may comprise a flag to indicate the nature of the transferred data (commands or other data) and their destination (memory or instruction register). The direct execution of transferred card commands provides an effective way of loading data onto a card, or of changing and/or creating data structures on a card.

After storing of otherwise processing the data in step **209**, the counter T is incremented in step **210**. Subsequently, in step **211**, the value of T is compared with the (expected) number of transfer commands. If all transfer commands have been processed, the routine is terminated in step **212**, and the state in which (preferably only) transfer commands are accepted is discontinued. Otherwise, the next transfer command is received in step **205**.

The sequence number S contained in the transfer commands serve several purposes. In the first place, they provide a mechanism to check the correct order in which the transfer commands are received and processed. In the second place, they provide a mechanism to resume an interrupted sequence of transfer commands. If for example a transmission error occurs and the routine of FIG. **5** is exited after, say, 5 out of 10 a series of transfer commands have been processed, the last sequence number S and/or the counter T can be used to resume the processing at the correct command when the series of commands is retransmitted. Also, the last sequence number S and/or the counter T can be used to undo the effect of the transfer commands which were processed. In order to undo the transfer commands of the failed series, a new series of commands may be provided which has the effect of deleting or undoing the results of the first series.

In the examples set out above, it is assumed that the card reproduces the authentication values R1, R2, etc. As this requires a certain amount of processing power and processing time, it is possible to simplify the procedure by using the authentication code of each preceding command as authentication value. Referring to FIG. **3**, this would imply that the value of R1 is chosen to be equal to MAC0, R2 is equal to MAC1, etc. The initial authentication value R0 may, in that case, be omitted. Such a scheme has the advantage of a greater speed and simplicity at the cost of a less effective security. In order to increase the level of security in the simplified procedure, it is possible not to use the (preceding) authentication code proper, but a value derived from the code as authentication value: R2=F(MAC1), where F is for example a hash or parity function.

A series of commands, as e.g. depicted in FIG. **3**, may be offered to one or more cards. In order to control the distribution of a specific series of commands, the initiation command (INIT) preferably comprises a distribution profile (DP) wich is compared with a card profile (CP) stored in the card. The initiation command is only executed if the distribution profile (DP) and the card profile (CP) match. The profiles may contain e.g. 16 bytes of user and/or card related information. The distribution profile contains information specific for a set of cards and may therefore contain wildcards which are replaced by corresponding information of the card profile before the profiles are compared. The term "match" should therefore be understood to include the corresponding of wildcards. The profiles can be used for only checking the authorization of the card by including in the service profile a flag indicating that the card is not to be put in the transfer commands accepting mode. By way of example it is assumed that each profile comprises 16 bits. The profiles:

CP=1001 0001 1010 0111 and

DP=1001 0001 1010 1001

match if the four least significant bits are considered as wildcards, i.e. are not taken into consideration. In other words, the distribution profile:

DP=1001 0001 1010 1001

"matches" all card profiles having 1001 0001 1010 as the first 12 digits. Cards having another card profile (CP) will not execute the initiation command and will thus not allow their data to be modified.

The method of the invention, as will be apparent from the above, involves basically three stages:

1. Producing a set of commands, i.e. one initiation command and as many transfer commands as required to transfer the data in question. This involves producing a set of interrelated authentication values. Each command is further provided with an authentication code (MAC).
2. Transmitting the set of commands to the card.
3. Executing the commands on the card after checking the authentication code of each command. The executing involves in the case of the transfer commands the checking of i.a. the authentication values and then the loading of the relevant data in memory or (in the case of a card command) in the instruction register of the processor of the card.

The method of the present invention thus enables the secure transfer of data, e.g. card commands, to a smart card by providing a double protection mechanism: the related authentication values guarantee that the sequence of

received data is unaltered, while the authentication codes accompanying each command provide protection of the transferred data. In the case where the transferred data are smart card commands, the method thus provides protection of non-protected commands.

A system for modifying data on a smart card is schematically shown by way of example in FIG. 6. The system comprises a card reader/writer 5 connected by a data link 6 to a computer 7. A smart card 1 is inserted in the card reader/writer 5. A set of commands, as e.g. represented in the table of FIG. 3, is prepared in the computer 7. Then the commands of the set are sent via the data link 6 to the card reader/writer 5, which transfers the commands to the smart card 1. The smart card executes the commands as described above, thus storing the relevant data in its memory.

It will be understood by those skilled in the art that the embodiments described above are given by way of example only and that many modifications and additions are possible without departing from the scope of the present invention.

We claim:

1. A method of securely modifying data in a smart card (1), the method comprising the steps of:

producing an initiation command (INIT) comprising an initial authentication value (R0) and producing an initial authentication code (MAC0) based on the command and the initial value (R0),

producing at least one transfer command (TRAN) comprising data (DATA) to be transferred and producing a subsequent authentication code (MAC1, MAC2, . . .) based on both the command (TRAN) and a subsequent authentication value (R1, R2, . . .) derived from the initial value (R0),

transferring the commands and their authentication codes thus produced to the smart card (1),

authenticating the commands in the smart card (1) by checking the authentication codes (MAC0, MAC1, . . .) and checking the subsequent authentication values (R1, . . .) derived from the initial value (R0), and

storing the transferred data (DATA) in the card.

2. A method according to claim 1, wherein the initiation command (INIT) further comprises the number of transfer commands (N).

3. A method according to claim 1, wherein the initiation command (INIT) is arranged to prohibit the card (1) from accepting commands other than transfer commands (TRAN).

4. A method according to claim 1, wherein each transfer command (TRAN) further comprises a sequence number (SN).

5. A method according to claim 1, wherein the transfer data (DATA) comprise a card command (e.g. UPDATE).

6. A method according to claim 5, wherein a card command (e.g. UPDATE) is directly executed upon transfer to the card.

7. A method according to claim 5, wherein a card command is a protected command according to the ETSI (European Telecommunications Standardization Institute) TE9-recommendation, the value needed for protecting the command corresponding with the authentication value.

8. A method according to claim 1, wherein the authentication codes (MAC1, MAC2, . . .) are generated according to the ANSI X9.19 standard.

9. A method according to claim 1, wherein deriving the subsequent values (R1, R2, . . .) involves the use of a secret key (K).

10. A method according to claim 1, wherein the initiation command (INIT) comprises a distribution profile (DP) and the card comprises a card profile (CP), and wherein the initiation command is only executed if the distribution profile (DP) and the card profile (CP) match.

11. A method according to claim 1, wherein the initiation command (INIT) puts the card (1) in a transfer state in which only transfer commands (TRAN) are executed.

12. A method according to claim 11, wherein the card (1) remains in the transfer state until it has received all transfer commands (TRAN), as indicated by the number (N) of transfer commands contained in the initiation command (INIT).

* * * * *

US006546549B2

(12) **United States Patent**
Li

(10) **Patent No.: US 6,546,549 B2**
(45) **Date of Patent: *Apr. 8, 2003**

(54) **SOURCE CODE TRANSFORMATION PROCESS AND RECORDING MEDIUM**

(75) Inventor: **Qiaoyun Li**, Saitama (JP)

(73) Assignee: **Sony Corporation**, Tokyo (JP)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/136,488**

(22) Filed: **Aug. 19, 1998**

(65) **Prior Publication Data**

US 2002/0013936 A1 Jan. 31, 2002

(30) **Foreign Application Priority Data**

Aug. 21, 1997 (JP) 9-225239

(51) **Int. Cl.**[7] **G06F 9/45**

(52) **U.S. Cl.** **717/137**; 717/108; 717/116; 717/136; 717/140; 717/147; 717/148

(58) **Field of Search** 717/136, 137, 717/108, 176, 116, 140, 147–148

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,999,988	A *	12/1999	Pelegri-Llopart et al.	709/304
6,058,431	A *	5/2000	Srisuresh et al.	709/245
6,157,961	A *	12/2000	Kessler et al.	709/315
6,347,342	B1 *	2/2002	Marcos et al.	709/315
6,453,362	B1 *	9/2002	Bittinger et al.	709/316

OTHER PUBLICATIONS

Stoyenko, "SUPRA–RPC: SUbprogram PaRAmeters in Remote Procedure Calls", IEEE, pp. 620–627, Dec. 1991.*
Bacon et al., "Using Events to Build Distributed Application", IEEE, pp. 148–155, Jun. 1995.*
Schmidt, "Evaluating Architectures For Multithreaded Object Request Brokers", ACM, pp. 54–60, Oct. 1998.*
OMG Document, "The Common Object Request Broker: Architecture and Specification", Digital Equipment Co., Hewlett–Packard Co., HyperDesk Co., NCR Co., Object Design, Inc., SunSoft, Inc. pp 1–178, Dec. 1993.*

* cited by examiner

Primary Examiner—Gregory Morse
Assistant Examiner—Ted T. Vo
(74) *Attorney, Agent, or Firm*—Frommer Lawrence & Haug LLP; William S. Frommer; Glenn F. Savit

(57) **ABSTRACT**

A process for transforming an original source code containing a description of a stub method employed in an object interaction into another source code corresponding to an environment on which a program is executed. The original source code is described with a predetermined programming language and contains information concerning the stub method to be used in the object interaction. The information is described in a format which is common to a plurality of different program execution environments. The transformation is conducted with reference to a registered information corresponding to the program execution environment on which stub method is to be executed into another source code described with the same programming language as the original source code in a predetermined format corresponding to the program execution environment on which the stub method is to be executed.

12 Claims, 4 Drawing Sheets

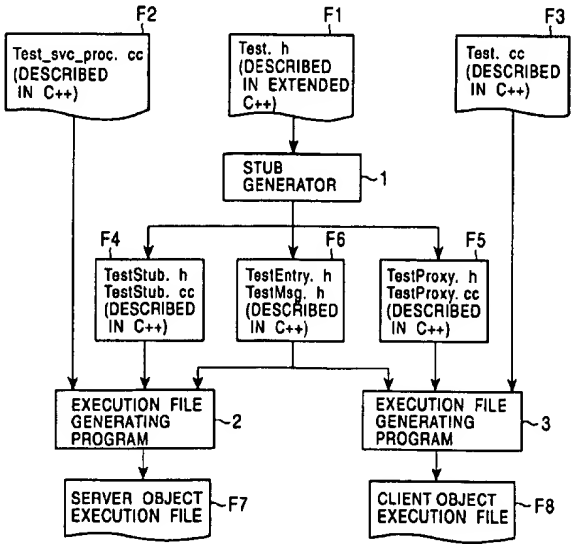


FIG. 1

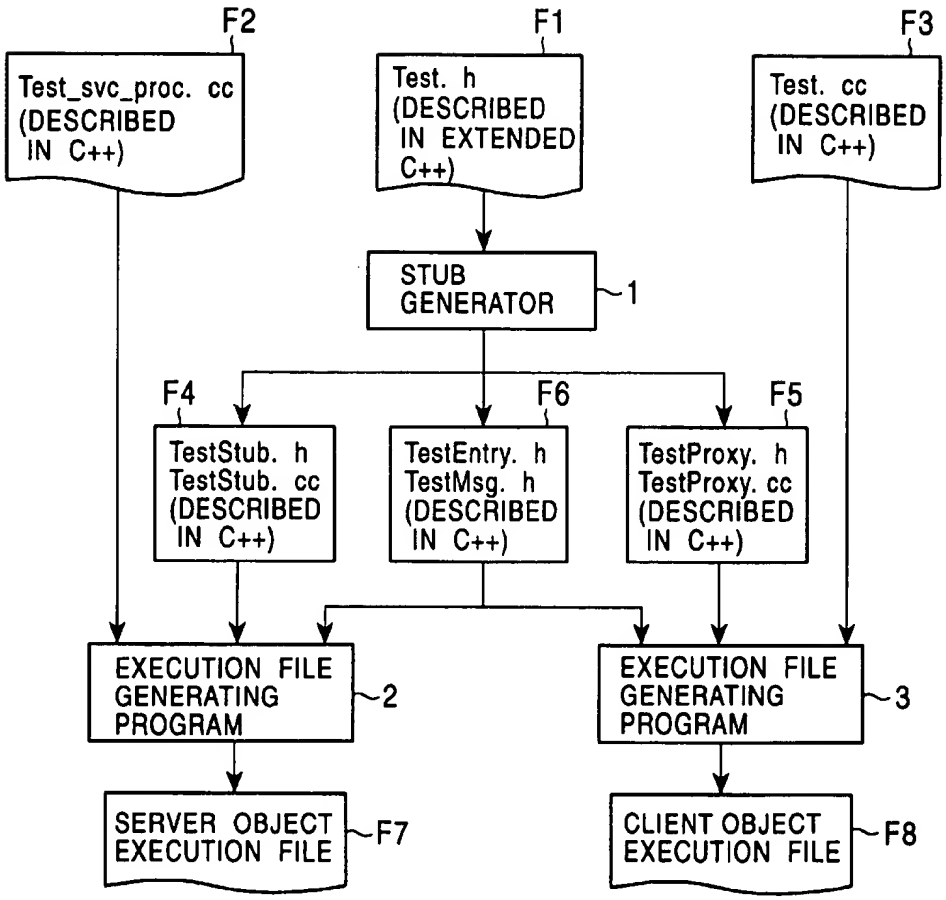

F2
F1
F3
Test_svc_proc. cc (DESCRIBED IN C++)
Test. h (DESCRIBED IN EXTENDED C++)
Test. cc (DESCRIBED IN C++)
STUB GENERATOR
1
F4
F6
F5
TestStub. h TestStub. cc (DESCRIBED IN C++)
TestEntry. h TestMsg. h (DESCRIBED IN C++)
TestProxy. h TestProxy. cc (DESCRIBED IN C++)
EXECUTION FILE GENERATING PROGRAM
2
EXECUTION FILE GENERATING PROGRAM
3
SERVER OBJECT EXECUTION FILE
F7
CLIENT OBJECT EXECUTION FILE
F8

FIG. 2

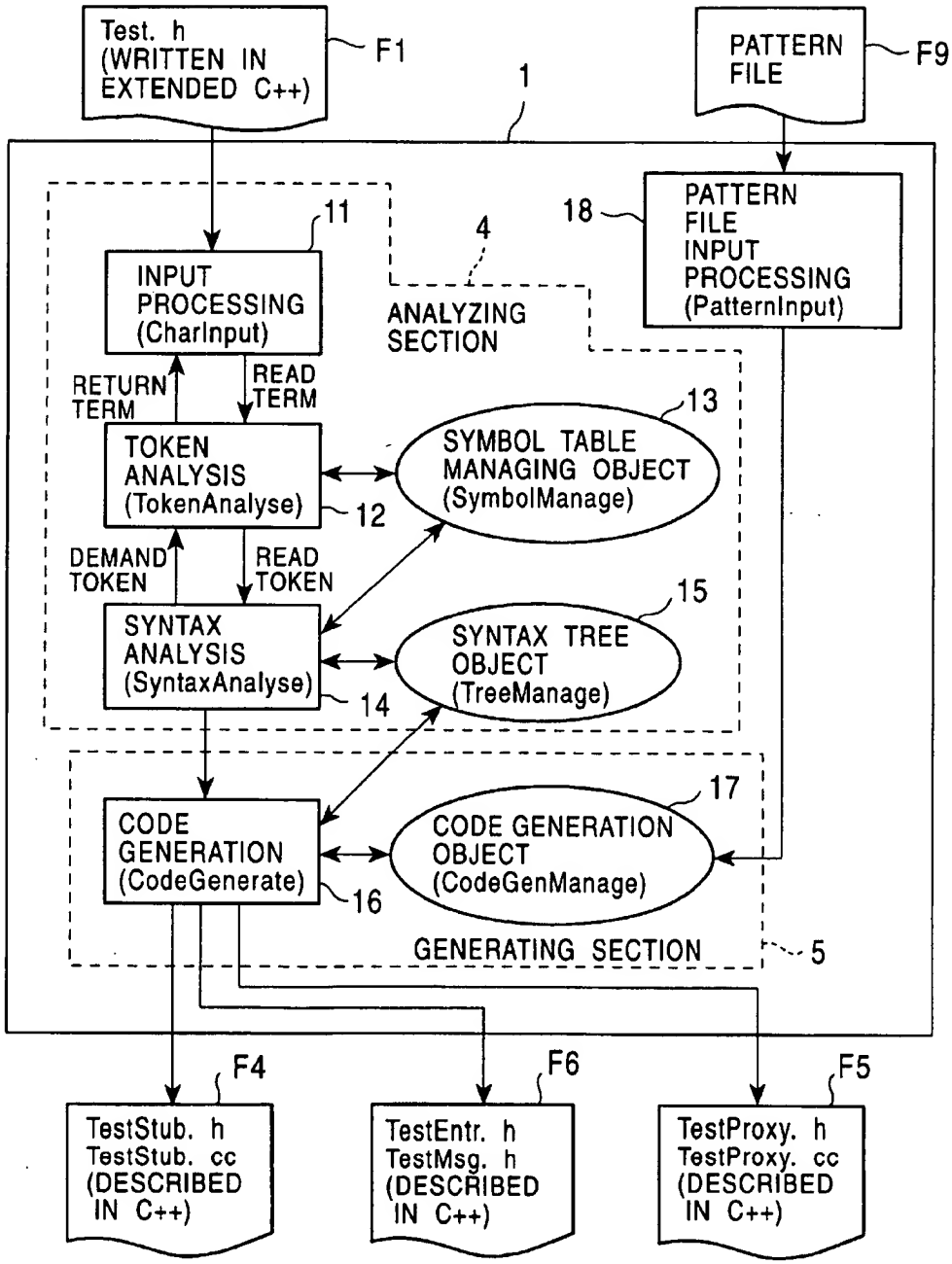

Test. h
(WRITTEN IN
EXTENDED C++)
F1
PATTERN
FILE
F9
1
11
4
18
PATTERN
FILE
INPUT
PROCESSING
(PatternInput)
INPUT
PROCESSING
(CharInput)
ANALYZING
SECTION
RETURN
TERM
READ
TERM
13
TOKEN
ANALYSIS
(TokenAnalyse)
12
SYMBOL TABLE
MANAGING OBJECT
(SymbolManage)
DEMAND
TOKEN
READ
TOKEN
15
SYNTAX
ANALYSIS
(SyntaxAnalyse)
14
SYNTAX TREE
OBJECT
(TreeManage)
17
CODE
GENERATION
(CodeGenerate)
16
CODE GENERATION
OBJECT
(CodeGenManage)
GENERATING SECTION
5
F4
F6
F5
TestStub. h
TestStub. cc
(DESCRIBED
IN C++)
TestEntr. h
TestMsg. h
(DESCRIBED
IN C++)
TestProxy. h
TestProxy. cc
(DESCRIBED
IN C++)

FIG. 3

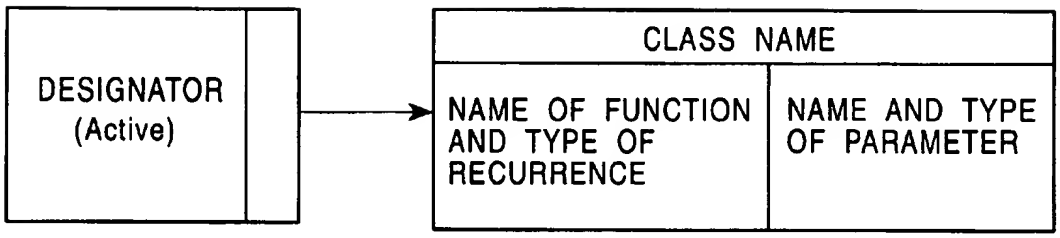
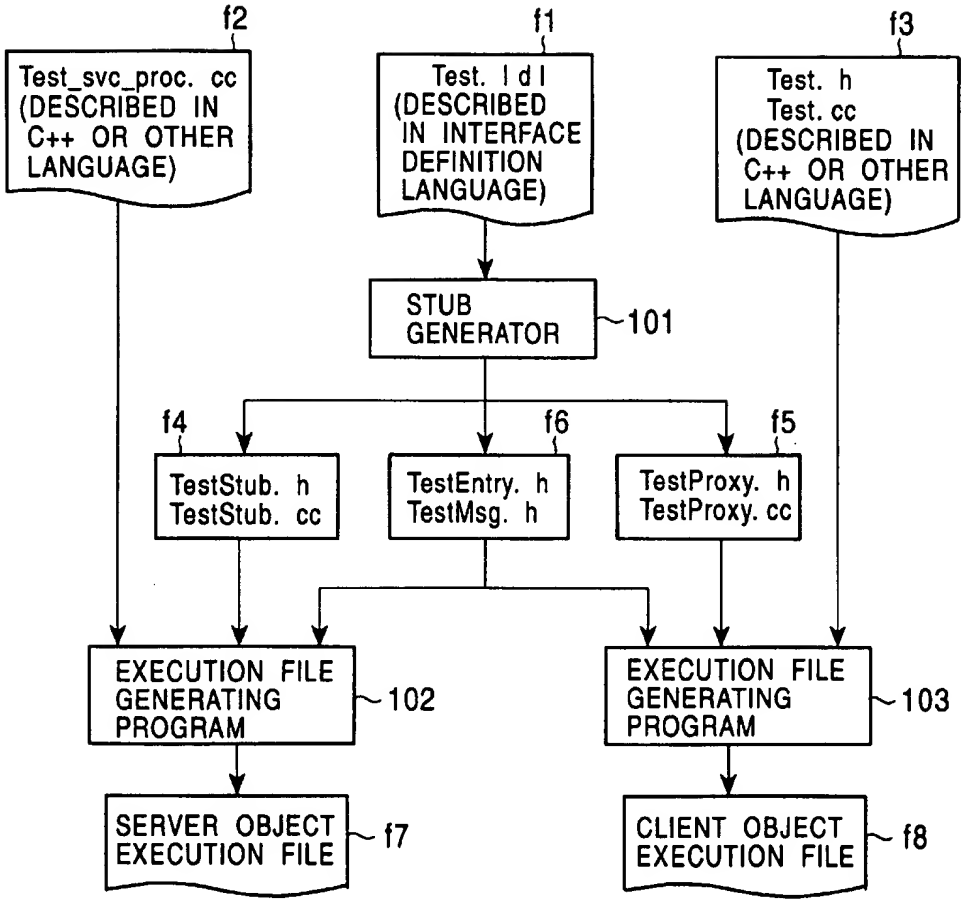

DESIGNATOR
(Active)
CLASS NAME
NAME OF FUNCTION AND TYPE OF RECURRENCE
NAME AND TYPE OF PARAMETER

FIG. 4


f2
Test_svc_proc. cc
(DESCRIBED IN C++ OR OTHER LANGUAGE)
f1
Test. I d l
(DESCRIBED IN INTERFACE DEFINITION LANGUAGE)
f3
Test. h
Test. cc
(DESCRIBED IN C++ OR OTHER LANGUAGE)
STUB GENERATOR
101
f4
f6
f5
TestStub. h
TestStub. cc
TestEntry. h
TestMsg. h
TestProxy. h
TestProxy. cc
EXECUTION FILE GENERATING PROGRAM
102
EXECUTION FILE GENERATING PROGRAM
103
SERVER OBJECT EXECUTION FILE
f7
CLIENT OBJECT EXECUTION FILE
f8

SOURCE CODE TRANSFORMATION PROCESS AND RECORDING MEDIUM

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a source code transformation process for transforming source code including a description concerning an object interaction method known as a stub method into source code which is adapted to a program execution environment. The present invention also is concerned with a computer-readable recording medium which stores a source code transformation program for implementing the source code transformation process. Details of certain features of the present invention are described in European Patent Application No. 0.753,811 A1 entitled "Data processing method and device" and filed by the same assignee on Jul. 12, 1996 claiming a Convention Priority on JP 178625/95, filed Jul. 14, 1997, the complete disclosure of which is hereby incorporated herein by reference.

2. Description of the Related Art

In general, object-oriented programming frequently uses object interaction methods. The term "object interaction" is used in this specification to mean, for example, a process for sending a message from one object to another. The object interaction is an essential service offered by an operating system in object-oriented programming.

Methods employed in object interaction are generally referred to as "stub methods". Thus, a stub method is used in object-oriented programming for the purpose of sending a message from one object to another. In the following description, the object from which the message is sent will be referred to as a "client object", while the object which receives the message will be referred to as a "server object".

For instance, when it is desired to execute the procedure of a server object on another computer linked through a network, a message is sent through the network from the client object to the server object to be executed on the other computer. It is thus possible to execute the procedure of the server object on the other computer linked through the network. It will be understood that a remote control operation for calling a procedure to be executed on a different computer through a network can be performed in the same way as that performed by a local procedure call interface.

In general, a stub generator is used for generating a stub method. More specifically, such a stub generator reads source code containing information concerning a stub method described in a certain programming language and generates a stub method from the read source code. In most cases, the stub generator has both a function for generating a stub method concerning the client object which is the message sender and a function for generating a stub method concerning the server object which is the message receiver. With these functions, the client object and the server object can be set up very easily with a high degree of versatility.

A description will be given of a process for forming a program by using a stub generator.

Formation of the program using a stub generator includes describing a program for a non-object-interaction portion by means of the C++ programming language and describing a program for an object-interaction portion by using an interface definition language. The "interface definition language" is a language which defines the interface for the communication between the objects. The interface definition language

may be, for example, a language determined by the CORBA (Common Object Request Broker Architecture) which is a rule for implementing object-oriented distributed processing environments.

More specifically, referring to FIG. **4**, the process of forming an application program for object interaction begins with the formation of three kinds of source codes which are referred to as the "first, "second" and "third" source codes.

The first source code contains stub method information described in the interface definition language. This means that the interface to be used for the object interaction is defined by the first source code. In FIG. **4**, a file **f1** in which the first source code is written is indicated as "Test.idl".

The second source code contains server object information described in a programming language such as C++. Thus, the second source code contains the portion of the server object information other than the object interaction information described in the first source code. The second source code further contains, for example, a procedure for calling, from the stub method described by the first source code, a method for generating a thread which performs processing based on the received message. In FIG. **4**, a file **f2** describing the second source code is represented by "Test svc_proc.cc".

The third source code contains the client object information described in a programming language such as C++. It is to be noted that the third source code includes the portion of the client object information other than the object interaction information described in the first source code. Thus, the third code contains a description for implementing a main function, e.g., connection to an appropriate service, in the form of an application program. In FIG. **4**, a file **f3** in which the third code is described is represented by "Test.cc" and "Test.h". The "Test.h" file is an include file which is incorporated in the "Test.cc" file.

The first source code described in the interface definition language is transformed by a stub generator **101** into source code described with the same programming language as that used for the description of the second and third source codes. Thus, the stub generator **101** generates a stub method which is described by a programming language such as C++ rather than by the interface definition language.

More specifically, the transformation performed by the stub generator **101** generates from the file **f1** in which the first code is described, a file **f4** which contains a stub method description for the server object, a file **f5** containing a stub method description for the client object, and an include file **f6** which is to be commonly used by objects involved in the object interaction.

In FIG. **4**, the file **f4** containing the stub method description for the server object is represented by "TestStub.cc" and "TestStub.h". The "TestStub.h" file is an include file incorporated in the "TestStub.cc" file. The file **f5** containing stub method description for the client object is represented by "TestProxy.cc" and "TestProxy.h". The "TestProxy.h" file is an include file incorporated in the "TestProxy.cc" file. The include file **f6** commonly used by the objects involved in the object interaction is represented by "TestEntry.h" and "TestMsg.h".

Then, compilation is performed on each of the file **f2** containing the description of the second source code, the file **f3** containing the description of the third source code, and the files **f4** to **f6** generated by the stub generator **101**, and the results of the compilation are linked, thereby forming a server object executable file **f7** for the server object and a client object executable file **f8**

Thus, in the process shown in FIG. 4, the server object executable file f7 is generated by an executable file generating program **102** having a compiler and a linker from the resources which include the file f2 (Test_svc_proc.cc) describing the second source code, the file f4 (TestStub.cc, TestStub.h) containing the stub method description for the server object, and the include file f6 (TestEntry.h, TestMsg.h). which is commonly used by the objects taking part in the object interaction. At the same time, the client object executable file f8 is generated by an executable file generating program **103** having a compiler and a linker from the resources which include the file f3 (Test.h, Test.cc) describing the third source code, the file f5 (TestProxy.cc, TestProxy.h) containing the stub method description for the client object, and the include file f6 (TestEntry.h, TestMsg.h). which is commonly used by the objects taking part in the object interaction.

In general, there are a variety of environments under which programs are executed. Any application program, therefore, has to be formed so as to be suited to the program execution environment. This naturally requires that the program execution environment is suited also to the stub method which is generated in accordance with the nature of the application program and the demand. The stub generator, therefore, has to generate the stub method such that the stub method is suited to the execution environment of the application program that relies upon the generated stub method.

Therefore, it has been a conventional practice to prepare different stub generators which generate stub methods for different program execution environments. Thus, generation of a specific stub method requires selective use of the stub generator in accordance with the environment in which the application program which uses the stub method is to be executed.

Thus, the conventional technique for generating a stub method employs an operation performed by a stub generator for transforming source code described in an interface definition language. The following problems are encountered by this known technique.

Firstly, studying and learning interface definition languages are essential in order for the programmer to form an application program which uses a stub method, because the formation of source code described in a certain interface definition language is necessary. In general, interface definition languages and programming languages such as C++ have fundamentally different formats. Thus, an ordinary programmer, even if familiar with programming languages, has to spend much time studying and learning interface definition languages to such a degree as to become able to form a source code with an interface definition language. Consequently, much time and labor are necessary for forming an application program which uses a stub method.

A second problem pertains to the difficulty that lies in ascertaining matching between the interface constituted by the source code portion described in the interface definition language and the interface corresponding to the source code portion described in a programming language such as C++. This difficulty will be described in detail. A single application program formed by the procedure shown in FIG. 4 contains both a portion described in the interface definition language and a portion described in the programming language such as C++. As stated before, these two types of languages employ entirely different formats. Therefore, co-existence of a description in the interface definition language and a description in the programming language such as C++ makes it extremely difficult to confirm whether or not the interfaces of these descriptions match with each other.

A third problem is that the conventional stub generator cannot be used commonly for a plurality of different program execution environments. Therefore, as stated before, is has been necessary to prepare a plurality of stub generators which generate different stub methods suited to different program execution environments.

Fourthly, it is to be pointed out that only methods which have been determined by the interface definition languages are usable as the stub method. Thus, the users are not allowed to interactively change the contents of the stub method. In other words, the stub methods are determined in the procedure for designing the stub generators and, hence, are dependent on the stub generators. The contents of the stub methods therefore cannot be altered by users.

SUMMARY OF THE INVENTION

Accordingly, it is an object of the present invention to provide a source code transformation process which can easily be adapted to a variety of program execution environments and which can generate stub methods without using any interface definition language.

Another object of the present invention is to provide a computer-readable recording medium storing a source transformation program which implements the above-described source code transformation process.

To these ends, according to one aspect of the present invention, there is provided a source code transformation process, comprising the steps of: preparing a primary source code described with a predetermined programming language and containing information concerning a method to be used in an object interaction, the information being described in a format which is common to a plurality of different program execution environments; and transforming, by making a reference to registered information corresponding to the program execution environment on which the method is to be executed, the primary source code into a secondary source code described with the same programming language as the primary source code in a predetermined format corresponding to the program execution environment on which the method is to be executed.

According to another aspect of the present invention, there is provided a computer-readable recording medium, storing a source code transformation program that implements a process comprising the steps of: preparing a primary source code described with a predetermined programming language and containing information concerning a method to be used in an object interaction, the information being described in a format which is common to a plurality of different program execution environments; and transforming, by making a reference to registered information corresponding to the program execution environment on which the method is to be executed, the primary source code into a secondary source code described with the same programming language as the primary source code in a predetermined format corresponding to the program execution environment on which the method is to be executed.

These and other objects, features and advantages of the present invention will become clear from the following description of the preferred embodiment taken with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is an illustration of a procedure for forming a stub method by means of a stub generator incorporating the present invention;

FIG. 2 is a diagram showing a process performed by the stub generator;

FIG. 3 is an illustration of a class definition conducted in accordance with a designator "Active"; and

FIG. 4 is an illustration of a conventional procedure for generating a stub method.

DESCRIPTION OF THE PREFERRED EMBODIMENT

A source transformation process in accordance with the present invention will be described through illustration of a procedure for forming an application program which implements object interaction between a client object and a server object. The recording medium in accordance with the present invention stores a program which implements a stub generator described later. The term "recording medium" is used to mean any type of computer-readable storage medium such as, for example, a magnetic disk, a magneto-optical disk, a phase-change optical disk, an optical disk, a ROM (Read Only Memory), a RAM (Random Access Memory), and so on.

Referring to FIG. 1, the process of forming an application program for object interaction begins with formation of three kinds of source codes which are referred to as the "first, "second" and "third" source codes.

The first source code contains a stub method description. This means that the interface to be used for the object interaction is defined by this first source code. The first source code is described with C++ extended to allow the use of a predetermined designator "Active" which is intended to enable the description concerning the stub method. The designator "Active" has been defined by an extension of C++ specifically for the purpose of carrying out the present invention. A member function designated as "Active" is the member function corresponding to the stub method. In FIG. 1, a file F1 in which the first source code is described is represented by "Test.h".

The second source code contains server object information described in C++. The second source code contains the portion of the server object information other than the object interaction information described in the first source code. The second source code further contains, for example, a procedure for calling, from the stub method described by the first source code, a method for generating a thread which performs a processing based on a received message. In FIG. 1, a file F2 describing the second source code is represented by "Test_svc_proc.cc".

The third source code contains client object information described in C++. It is to be noted that the third source code includes the portion of the client object information other than the object interaction information described in the first source code. Thus, the third code contains a description for implementing a main function, e.g., connection to an appropriate service, in the form of an application program. In FIG. 1, a file F3 in which the third code is described is represented by "Test.cc".

The first source code is transformed by a stub generator 1 into a source code described with C++ without the use of the above-mentioned designator "Active". In other words, the stub generator 1 transforms the source code, such that the member function designated by the designator "Active" is described with C++ that has not been extended to enable the use of the designator "Active". Consequently, a stub method is generated which is described in the same programming language, i.e., C++, as those used for the description of the second and third source codes.

More specifically, the transformation performed by the stub generator 1 generates from the file F1 in which the first code is described a file F4 which contains a stub method description for the server object, a file F5 containing a stub method description for the client object, and an include file F6 which is to be commonly used by objects involved in the object interaction.

In FIG. 1, the file F4 containing the stub method description for the server object is represented by "TestStub.cc" and "TestStub.h". This file F4 will be hereinafter referred to as a "server stub file F4". The "TestStub.h" file is an include file incorporated in the "TestStub.cc" file. The file F5 containing the stub method description for the client object is represented by "TestProxy.cc" and "TestProxy.h". This file F5 will be hereinafter referred to as a "client stub file F5". The "TestProxy.h" file is an include file incorporated in the "TestProxy.cc" file. The include file F6 commonly used by the objects involved in the object interaction is represented by "TestEntry.h" and "TestMsg.h". This file F6 will be referred to as a "common stub file F6". As a result of the described transformation, the source codes contained in the files F4, F5 and F6 are described with C++ as in the cases of the second and third source codes, without using the designator "Active".

Then, compilation is performed on each of the file F2 containing the description of the second source code, the file F3 containing the description of the third source code, and the files F4 to F6 generated by the stub generator 1, and the results of the compilation are linked, thereby forming a server object executable file F7 and a client object executable file F8.

Thus, in the process shown in FIG. 1, the server object executable file F7 is generated by an executable file generating program 2 having a compiler and a linker from the resources which include the file F2 (Test_svc_proc.cc) describing the second source code, the server stub file F4 (TestStub.cc, TestStub.h) containing the stub method description for the server object, and the common stub file F6 (TestEntry.h, TestMsg.h). which is commonly used by the objects taking part in the object interaction.

At the same time, the client object executable file F8 is generated by an executable file generating program 3 having a compiler and a linker from the resources which include the file F3 (Test.cc) describing the third source code, the client stub file F5 (TestProxy.cc, TestProxy.h) containing the stub method description for the client object, and the common stub file F6 (TestEntry.h, TestMsg.h) which is commonly used by the objects taking part in the object interaction.

A detailed description will now be given of the stub generator 1.

The stub generator 1 is designed to generate a stub method very easily and with a high degree of versatility without using any interface definition language, by virtue of the use of the designator "Active".

More specifically, the stub generator 1 is a source code transformation program which is formed by using an object-oriented programming technique, and is intended for generating a stub method to be employed in object interaction. The stub generator 1 performs transformation of source code which describes a stub method with the help of the designator "Active" into source code which describes the stub method in the C++ language without using such a designator. More specifically, in the procedure shown in FIG. 1, the stub generator 1 serves to transform the file F1 (Test.h) describing the stub method with the help of the designator "Active" so as to generate the server stub file F4 (TestStub.h,

TestStub.cc), the client stub file F5 (TestProxy.h, TestProxy.cc) and the common stub file F6 (TestEntry.h, TestMsg.h).

As will be seen from FIG. 2, the stub generator 1 is composed mainly of an analyzing section 4 which analyses an input source code, and a generating section 5 which generates a stub method from the analyzed source code. The analyzing section 4 divides the input source code into constituent elements to generate an intermediate expression of the program, while the generating section 5 generates a stub method from the intermediate expression generated by the analyzing section 4.

The configuration of the analyzing section 4 corresponds to the programming language which describes the input source code. Thus, the analyzing section 4 has a different configuration when a different programming language is used for describing the source code input to the stub generator 1. It is to be understood, however, that the analyzing section 4 generates the same intermediate expression regardless of the type of the programming language. A common generating section 5, therefore, can be used for different programming languages used for describing the source code input to the stub generator 1.

FIG. 2 shows, by way of example, the process performed by the stub generator 1. An input processing routine 11 (CharInput) receives the file F1 (Test.h) described with the C++ extended to use the designator "Active". The routine 11 removes null spaces and comments from the input source code, so as to extract meaningful programming terms. In this input processing routine 11, the designator "Active" which is defined by extended C++ is also recognized as a reserved word.

A subsequent routine, i.e., a term analyzing routine (TokenAnalyze) 12, analyzes the terms extracted in the input processing routine 11 and produces a token stream in which meaningful programming terms are arranged according to tokens which are units of logic. This term analyzing routine also recognizes the designator "Active" that is defined by the extended C++.

The input processing routine 11, as well as the term analyzing routine 12, uses a symbol table managing object (SymbolManage) 13. The symbol table managing object 13 has a data architecture referred to as a "symbol table" which contains information concerning various constituent elements of the program. For instance, the symbol table stores, as shown in FIG. 3, the information concerning the class designated by the designator "Active", including the name of the class and the name of functions contained in the class and their return values, as well as the names of the parameters of the functions and the types of the parameters. Each function may employ two or more parameters. In such a case, information registered in the table contains the names and types of these parameters.

The contents registered in the symbol table managing object 13 are dynamically changeable. That is to say, the contents can be altered as desired, in accordance with the stub method to be generated.

Referring back to FIG. 2, a syntax analysis (SyntaxAnalyse) 14 is then executed to perform a syntax analysis on the token stream produced as a result of the term analyzing routine 12. As a result of this analysis, so-called syntax tree information, indicative of the correlation between the tokens constituting the token stream, is generated and stored in the syntax tree object (TreeManage) 15. The syntax tree information thus obtained constitutes part of the intermediate expression described above.

The input source code is analyzed by the analyzing section 4 in the described manner, whereby an intermediate expression is obtained. Then, the generating section 5 operates to generate the target source code from the intermediate expression produced by the analyzing section 4. More specifically, a code generating routine (CodeGenerate) 16, which is included in the generating section 5 generates and outputs the target source code, based on the syntax tree information obtained as the result of the analysis performed by the syntax analyzing routine 14. More specifically, the code generating routine 16 calls a code generating object (CodeGenManage) 17. The code generating object 17 includes a method for generating the target source code from the intermediate expression obtained through the analysis performed by the analyzing section 4. Thus, the target source code is generated by the code generating object 17.

More specifically, the code generating object 17 generates and outputs the server stub file F4 (TestStub.h, TestStub.cc), the client stub file F5 (TestProxy.h, TestProxy.cc) and the common stub file F6 (TestEntry.h, TestMsg.h). These files F4, F5 and F6 have been described in C++, without the use of the designator "Active".

The stub generator 1 is configured so as to register predetermined pattern files F9 in the code generating object 17. These pattern files F9 are switchable so that stub methods corresponding to different program executing environments are generated, as will be understood from the following description.

In general, an application program is executed under a predetermined program executing environment which is provided by the operating system. There are a variety of program executing environments.

An operating system "Aperios" (registered trademark), for example, is designed to simultaneously provide a plurality of different program executing environments. Each of the program executing environments is referred to as a "meta space". More specifically, the operating system "Aperios" can simultaneously provide a meta space referred to as an "mAV meta space" and a meta space referred to as an "mCOOP meta space". The mAV meta space is for executing so-called proedural programs. An application program to be executed on the mAV meta space is described as a single object. On the other hand, the mCOOP meta space is used for object-oriented programs. In general, an application program to be executed on the meta space is constituted by a plurality of objects.

When the operating system "Aperios" is loaded on a device having a television (TV) function, the arrangement may be such that an application program for displaying moving images on the television is executed in the mAV meta space, while an application program, which implements a graphical user interface (GUI) for controlling an operation panel through which operation instructions are input to the device, is executed on the mCOOP meta space. The stub method employed in the application program executed on the mAV meta space is different from the stub method employed by the application program executed on the mCOOP meta space.

It is necessary that the stub method generated by the stub generator 1 suits the program execution environment on which the stub method is to be executed. More specifically, the stub method of the application program to be executed on the mAV space has to correctly function on the mAV meta space. Similarly, the stub method of the application program to be executed on the mCOOP meta space has to correctly function on the mCOOP meta space.

In the stub generator **1** used in the described embodiment, therefore, pattern files F9 corresponding to a plurality of different program execution environments can be registered in the code generating object **17**, in order to make it possible to generate stub methods for a plurality of different program execution environments. In other words, pattern files F9 corresponding to the respective program execution environments are registered in the code generating object **17** for selective use in accordance with the program execution environment on which the desired stub method is to be executed, whereby the stub method is generated in accordance with each of a plurality of different program execution environments.

More specifically, referring to FIG. **2**, pattern files F9 describing information necessary for generating stub methods corresponding to the program execution environments are formed and registered in the stub generator **1**, prior to the generation of a stub method. Upon receipt of the pattern files F9, the stub generator **1** activates a pattern file input processing routine (PatternInput) **18** which accepts these pattern files F9 and registers the accepted pattern files F9 as internal information in the code generating object **17**. It is to be noted that the registration of the pattern files F9 in the code generating object **17** is conducted with clear indication of the relations between the pattern files and the program execution environments, i.e., to which one of the program execution environments each pattern file corresponds.

The stub generator 1, when generating a stub method, makes a reference to one of the pattern files F9 that corresponds to the program execution environment on which the stub method to be generated is executed. Thus, a single stub generator **1** can generate stub methods corresponding to a plurality of different program execution environments.

It is also to be understood that the registration of the pattern files F9 for different program execution environments enables the stub generator **1** to dynamically alter the procedure for generating the stub method. For instance, when it is desired to generate a stub method corresponding to a new program execution environment, a pattern file F9 corresponding to the new program execution environment is input to the stub generator **1**, so that the stub generator **1** generates a stub method corresponding to the new program execution environment. Similarly, when an existing program execution environment has been updated, the pattern file F9 corresponding to this program execution environment is modified correspondingly and registered in place of the existing pattern file F9. Consequently, the stub method is generated for the updated program execution environment.

The source code inputted to the stub generator **1** is described by using the designator "Active" in a format which is common to different program execution environments, i.e., with no dependency on the program execution environment on which the stub method to be generated is executed. Thus, the contents described in each pattern file F9 include the information which is necessary for transforming the source code described in the format common to a plurality of different program execution environments into another source code corresponding to a selected program execution environment.

Thus, the stub generator **1** used in this embodiment transforms, by referring to the registered pattern file F9, the source code containing information of a stub method described in a format common to a plurality of different program execution environments into another source code of a predetermined format corresponding to the program execution environment on which the stub method is to be executed.

As will be understood from the foregoing description, the present invention makes it possible to generate a stub method without the use of an interface definition language, by virtue of the use of the stub generator **1**. This eliminates the necessity for the studying and learning of interface definition language which otherwise are necessary for when those who are not familiar with interface definition languages attempt to form an application program that uses a stub method. Thus, the present invention greatly facilitates formation of an application program that uses a stub method.

The stub generator 1 employed in the described embodiment, which enables generation of a stub method without the help of an interface definition language as stated above, provides another advantage in that it eliminates the problem encountered by the known art in regard to the difficulty in confirming coincidence between the interface presented by the portion described in an interface definition language and the interface of the portion described in a programming language such as C++. Thus, still another advantage offered by the use of this stub generator **1** is that the work performed by the stub method in regard to confirmation of the object interaction interface is greatly facilitated.

A further advantage is that, since a reference is made to a registered pattern file F9 corresponding to the program execution environment on which the stub method is to be executed, the transformation of the source code can be conducted by using a single stub generator **1** commonly for a plurality of different program execution environments.

A further advantage is that the stub generator **1** permits dynamic alteration of the stub method generating procedure, by virtue of the feature of registering a pattern file F9 corresponding to each of a plurality of different program execution environments. Therefore, the stub generator **1** can generate, by altering the contents of a pattern file F9, a stub method which was not contemplated in the initial design of the stub generator **1**.

As will be understood from the foregoing description, the present invention provides a source code transformation process which enables generation of a stub method with high degrees of ease and versatility, as well as wide adaptability to a variety of program execution environments, without using any interface definition language.

Although the invention has been described through its preferred form, it is to be understood that the described embodiment is only illustrative. For instance although the C++ has been specifically mentioned as the programming language which describes the source code, the present invention can equally be carried equally well by using another programming language. Further changes and modifications may be imparted thereto without departing from the scope of the present invention which is limited solely by the appended claims.

What is claimed is:

1. A source code transformation process, comprising the steps of:

inputting, as a first file, primary source code described with a first programming language which is both a non-interface-definition language and an extended version of a second programming language, said source code containing information concerning a stub method to be used in an object interaction, said information being described in a format which is common to a plurality of different program execution environments;

inputting a server object information source code as a second file containing a portion of server object infor-

mation other than object interaction information described in said primary source code;

inputting a client object information source code as a third file containing a portion of client object information other than object interaction information described in said primary source code;

transforming, by referring to registered information corresponding to the program execution environment on which said method is to be executed, said primary source code into a secondary source code described with the second programming language in a predetermined format corresponding to the program execution environment on which said method is to be executed, wherein said transformation generates a fourth file which contains a stub method description for a server object, a fifth file containing a stub method description for a client object, and an include file which is to be commonly used by objects involved in the object interaction; and

compiling said second, third, fourth, fifth and include files, and linking the results of said compilations, to thereby form a server object executable file and a client object executable file.

2. The source code transformation process of claim 1, wherein each of said program execution environments is a respective type of meta space.

3. The source code transformation process of claim 1, wherein said plurality of different program execution environments include a first meta space for executing procedural programs and a second meta space used for object-oriented programs.

4. The source code transformation process of claim 1, wherein said first programming language is extended C++ and said second programming language is C++.

5. A computer-readable recording medium, storing a source code transformation program that implements a process comprising the steps of:

inputting, as a first file, primary source code described with a first programming language which is both a non-interface-definition language and an extended version of a second programming language, said source code containing information concerning a stub method to be used in an object interaction, said information being described in a format which is common to a plurality of different program execution environments;

inputting a server object information source code as a second file containing a portion of server object information other than object interaction information described in said primary source code;

inputting a client object information source code as a third file containing a portion of client object information other than object interaction information described in said primary source code;

transforming, by referring to registered information corresponding to the program execution environment on which said method is to be executed, said primary source code into a secondary source code described with the second programming language in a predetermined format corresponding to the program execution environment on which said method is to be executed, wherein said transformation generates a fourth file which contains a stub method description for a server object, a fifth file containing a stub method description for a client object, and an include file which is to be commonly used by objects involved in the object interaction; and

compiling said second, third, fourth, fifth and include files, and linking the results of said compilations, to thereby form a server object executable file and a client object executable file.

6. The recording medium of claim 5, wherein each of said program execution environments is a respective type of meta space.

7. The recording medium of claim 5, wherein said plurality of different program execution environments include a first meta space for executing procedural programs and a second meta space used for object-oriented programs.

8. The recording medium of claim 5, wherein said first programming language is extended C++ and said second programming language is C++.

9. A computer-readable recording medium, storing a program that implements a source code transformation process for generating stub methods used in object interactions, said process comprising the steps of:

receiving, by a stub generator, a plurality of pattern files each corresponding to one of a plurality of program execution environments including at least a first meta space for executing procedural programs and a second meta space for executing object oriented programs;

registering said pattern files as internal information of said stub generator;

inputting a primary source code described with a first programming language which is an extended version of a second programming language, said first and second programming languages each being non-interface-definition languages, said source code containing information concerning a stub method to be used in an object interaction, and said information being described in a format which is common to a plurality of different program execution environments; and

transforming, by said stub generator referring to one of said registered pattern files corresponding to the program execution environment on which said stub method is to be executed, said primary source code into a secondary source code described with the second programming language in a predetermined format corresponding to the program execution environment on which said stub method is to be executed;

whereby said stub generator generates a plurality of stub methods corresponding to a plurality of program execution environments, respectively.

10. A source code transformation process for generating stub methods used in object interactions, comprising the steps of:

receiving, by a stub generator, a plurality of pattern files each corresponding to one of a plurality of program execution environments including at least a first meta space for executing procedural programs and a second meta space for executing object oriented programs;

registering said pattern files as internal information of said stub generator;

inputting a primary source code described with a first programming language which is an extended version of a second programming language, said first and second programming languages each being non-interface-definition languages, said source code containing information concerning a stub method to be used in an object interaction, and said information being described in a format which is common to a plurality of different program execution environments; and

transforming, by said stub generator referring to one of said registered pattern files corresponding to the pro-

gram execution environment on which said stub method is to be executed, said primary source code into a secondary source code described with the second programming language in a predetermined format corresponding to the program execution environment on which said stub method is to be executed;

whereby said stub generator generates a plurality of stub methods corresponding to a plurality of program execution environments, respectively.

11. The source code transformation process of claim **10**, wherein said first programming language is extended C++ and said second programming language is C++.

12. The source code transformation process of claim **10** wherein said transformation generates a fourth file which contains a stub method description for a server object, a fifth file containing a stub method description for a client object, and an include file which is to be commonly used by objects involved in the object interaction, and said process further comprising the steps of:

inputting a server object information source code as a second file containing a portion of server object information other than object interaction information described in said primary source code;

inputting a client object information source code as a third file containing a portion of client object information other than object interaction information described in said primary source code;

compiling said second, third, fourth, fifth and include files, and linking the results of said compilations, to thereby form a server object executable file and a client object executable file.

* * * * *

US005923884A

United States Patent [19]

Peyret et al.

[11] Patent Number: 5,923,884

[45] Date of Patent: Jul. 13, 1999

[54] SYSTEM AND METHOD FOR LOADING APPLICATIONS ONTO A SMART CARD

[75] Inventors: Patrice Peyret, Mountain View, Calif.; Gilles Lisimaque, Potomac, Md.

[73] Assignee: Gemplus S.C.A., France

[21] Appl. No.: 08/706,396

[22] Filed: Aug. 30, 1996

[51] Int. Cl.[6] G06F 13/00; G06F 15/16; G06K 7/01

[52] U.S. Cl. 395/712; 395/188.01; 235/382

[58] Field of Search 707/1; 380/20; 235/382, 382.5; 340/825.34; 395/712, 186, 188.01

[56] References Cited

U.S. PATENT DOCUMENTS

5,293,424	3/1994	Holtey et al.	235/382
5,509,073	4/1996	Monnin	380/20
5,588,146	12/1996	Leroux	707/1
5,826,011	10/1998	Chou et al.	395/186

Primary Examiner—Mark H. Rinehart

Attorney, Agent, or Firm—Gray Cary Ware & Freidenrich

[57] ABSTRACT

A system for loading an applet and its associated use rights into a smart card having other applets with associated use rights with values that change as the application is used is provided that stores, remotely from said smart card, an applet and use rights with a predetermined initial value, associated with the applet, and has a smart card having a processing unit, and a memory unit, the memory unit being connected to the processing unit and storing a second application having use rights. The smart card may be connected to said remote storage means, and the application, having use rights with a predetermined value, may be loaded from said remote storage means into said smart card. A smart card is also provided having a processor for executing an application, a memory, connected to the processor, for storing multiple applications, including a first application having first use rights and having first values associated with the first use rights, the first value changing from a predetermined initial value with use of the first use rights, a system for loading in the smart card a second application from a remote location over an interface, the second application having second use rights, a system for storing said second application into said memory in said smart card, and a system for changing the use rights of said first application and said second application. A method of replenishing the use rights in a smart card is also provided.

12 Claims, 7 Drawing Sheets

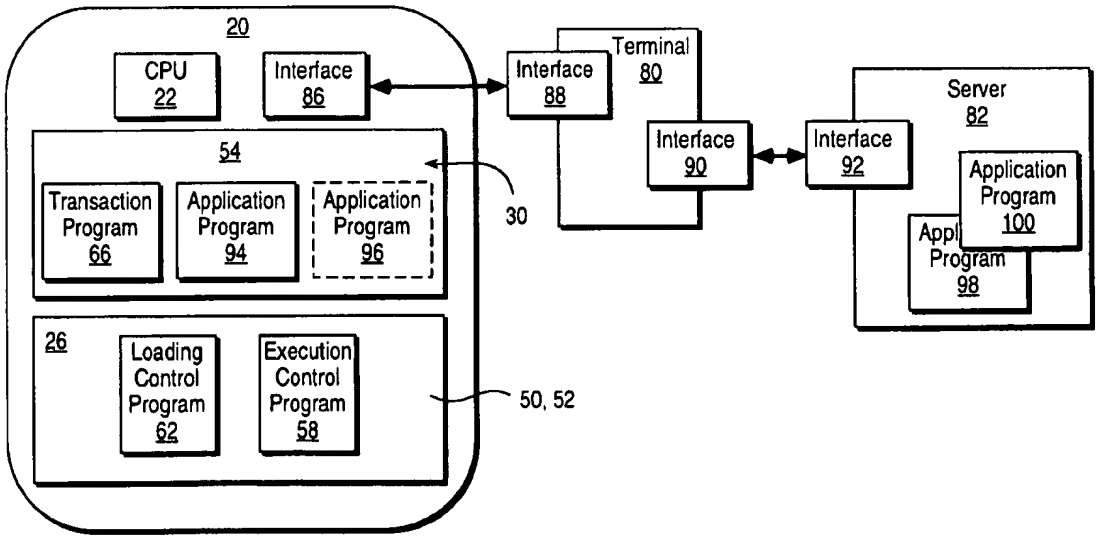


FIG. 1

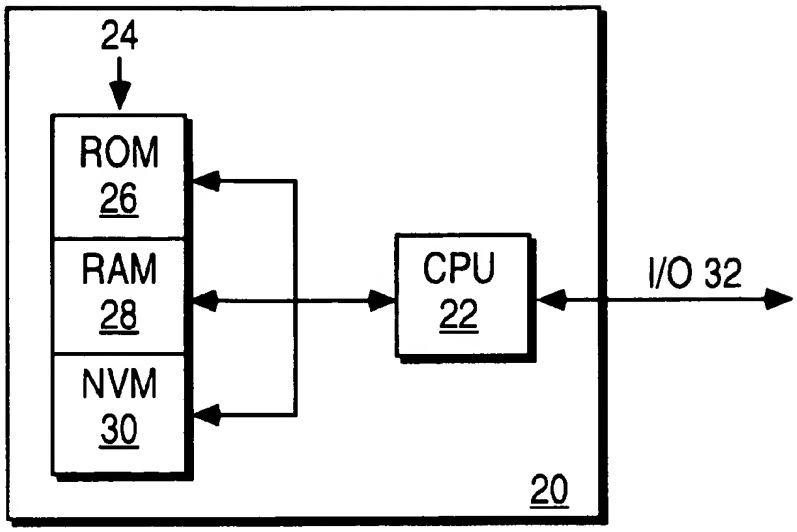

24
ROM
26
RAM
28
NVM
30
CPU
22
I/O 32
20

FIG. 2

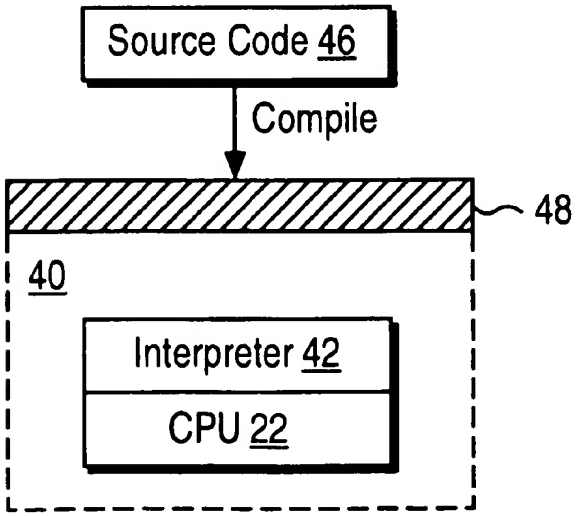
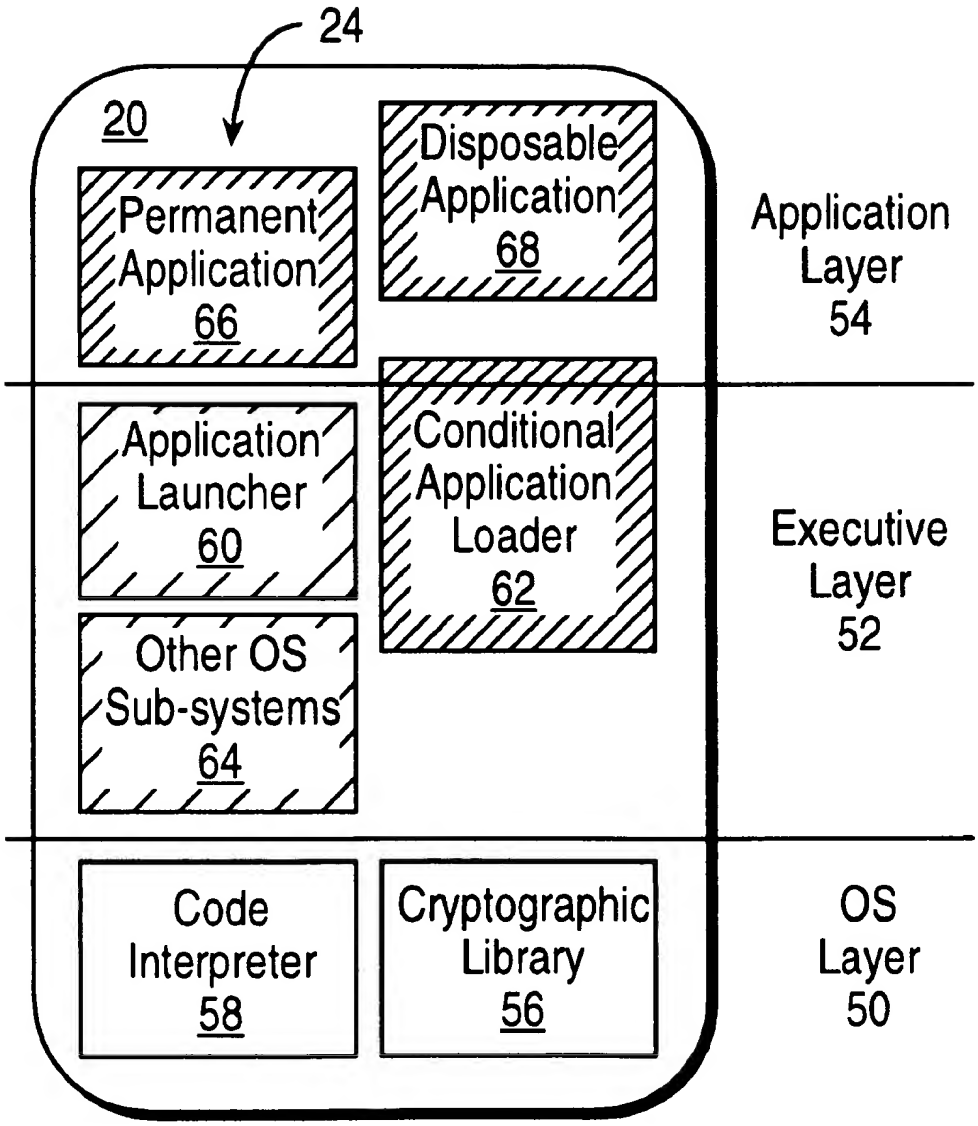

Source Code 46
Compile
48
40
Interpreter 42
CPU 22

FIG. 3


24
20
Permanent Application 66
Disposable Application 68
Application Launcher 60
Conditional Application Loader 62
Other OS Sub-systems 64
Code Interpreter 58
Cryptographic Library 56
Application Layer 54
Executive Layer 52
OS Layer 50

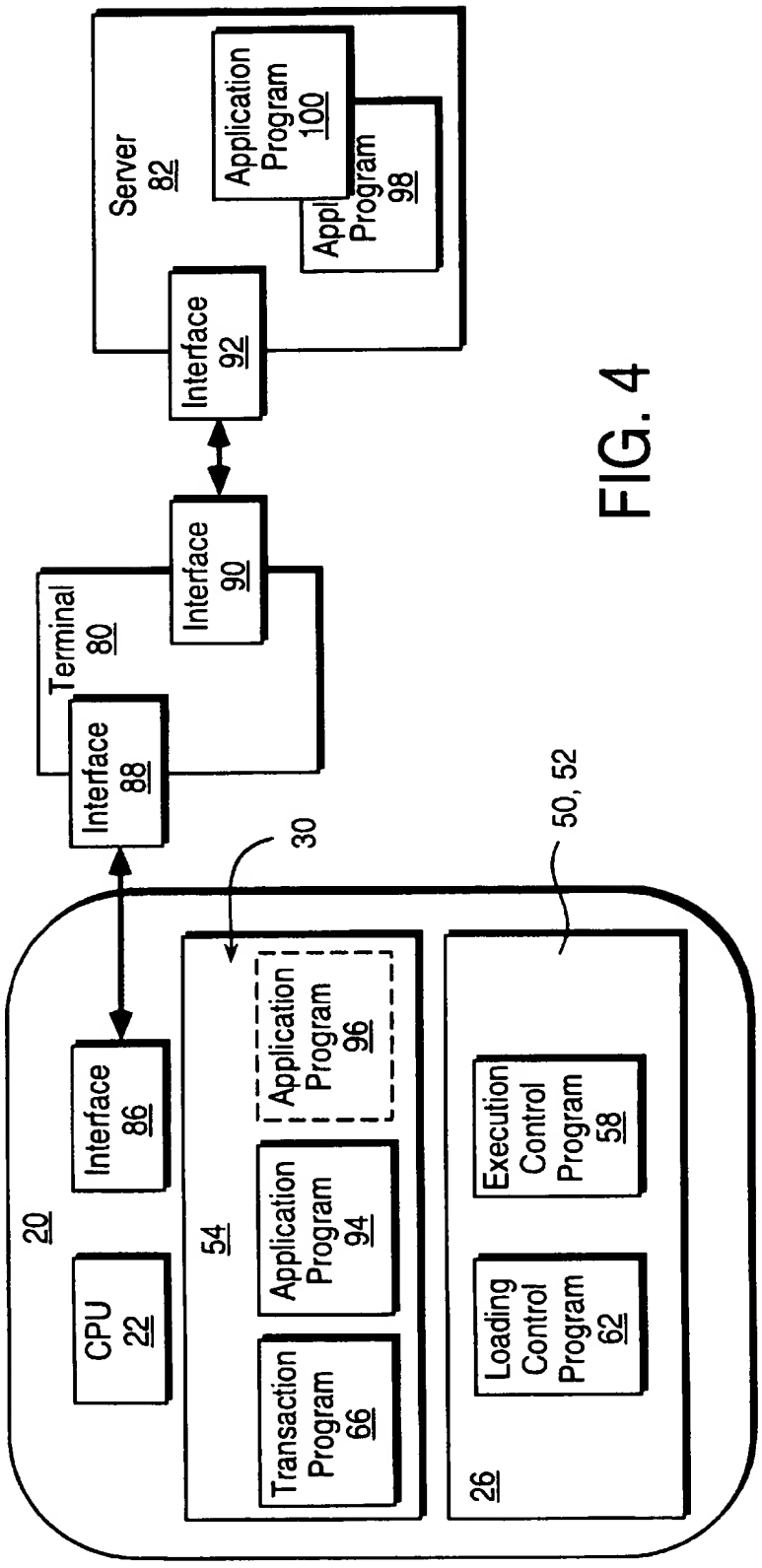

Server
82
Application Program
100
Appl Program
98
Interface
92
Interface
90
Terminal
80
Interface
88
30
50, 52
Interface
86
Application Program
96
20
54
Application Program
94
CPU
22
Transaction Program
66
Execution Control Program
58
Loading Control Program
62
26

FIG. 4

FIG. 5

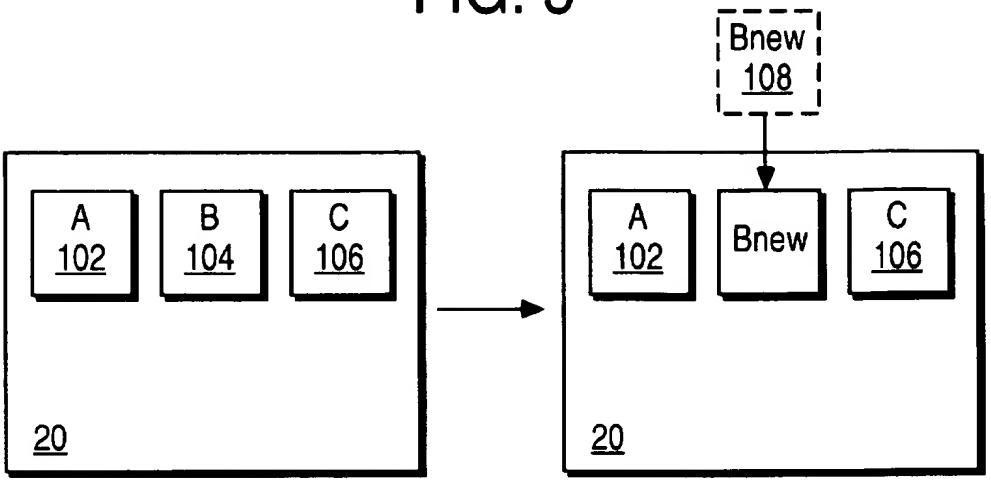
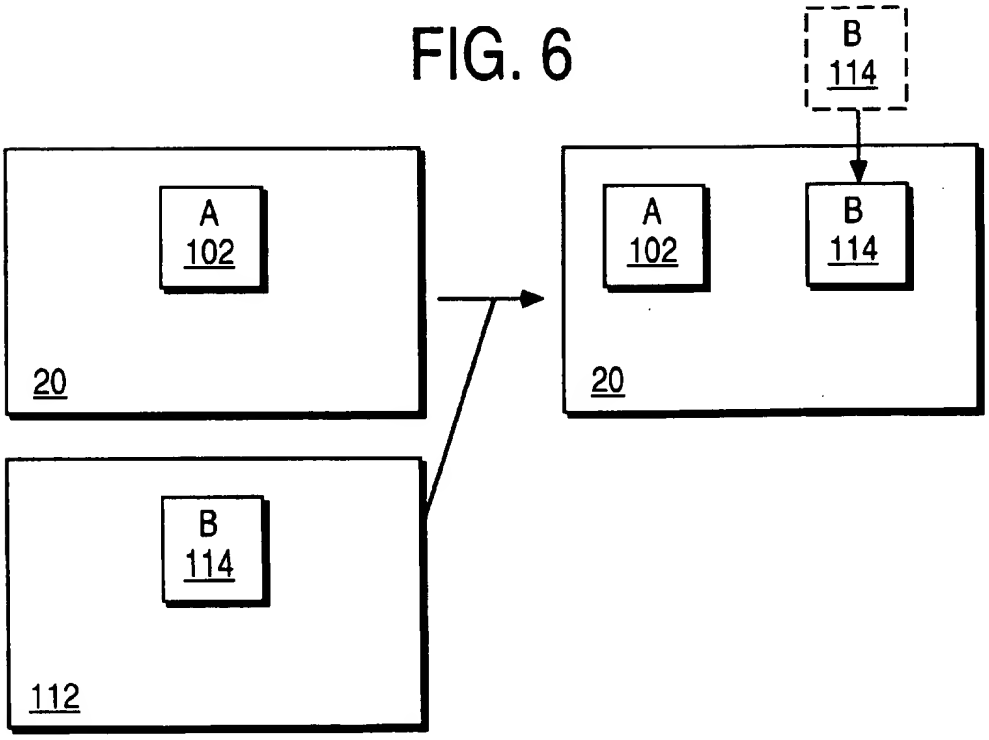

Bnew
108
A
102
B
104
C
106
20
A
102
Bnew
C
106
20

FIG. 6


B
114
A
102
20
B
114
112
A
102
B
114
20

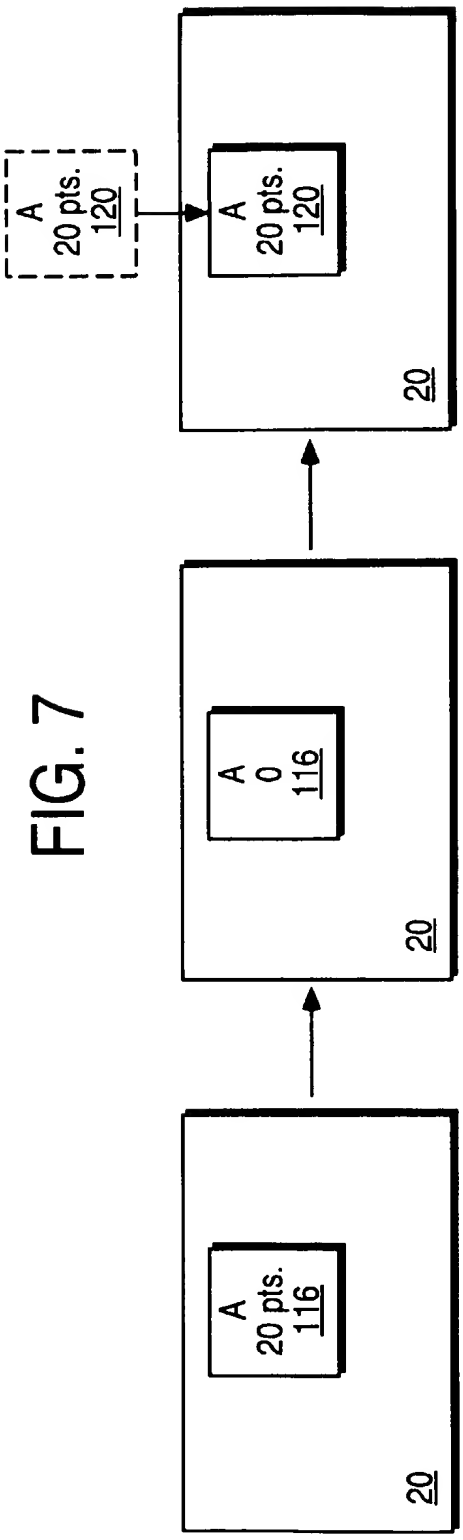


FIG. 7

FIG. 8

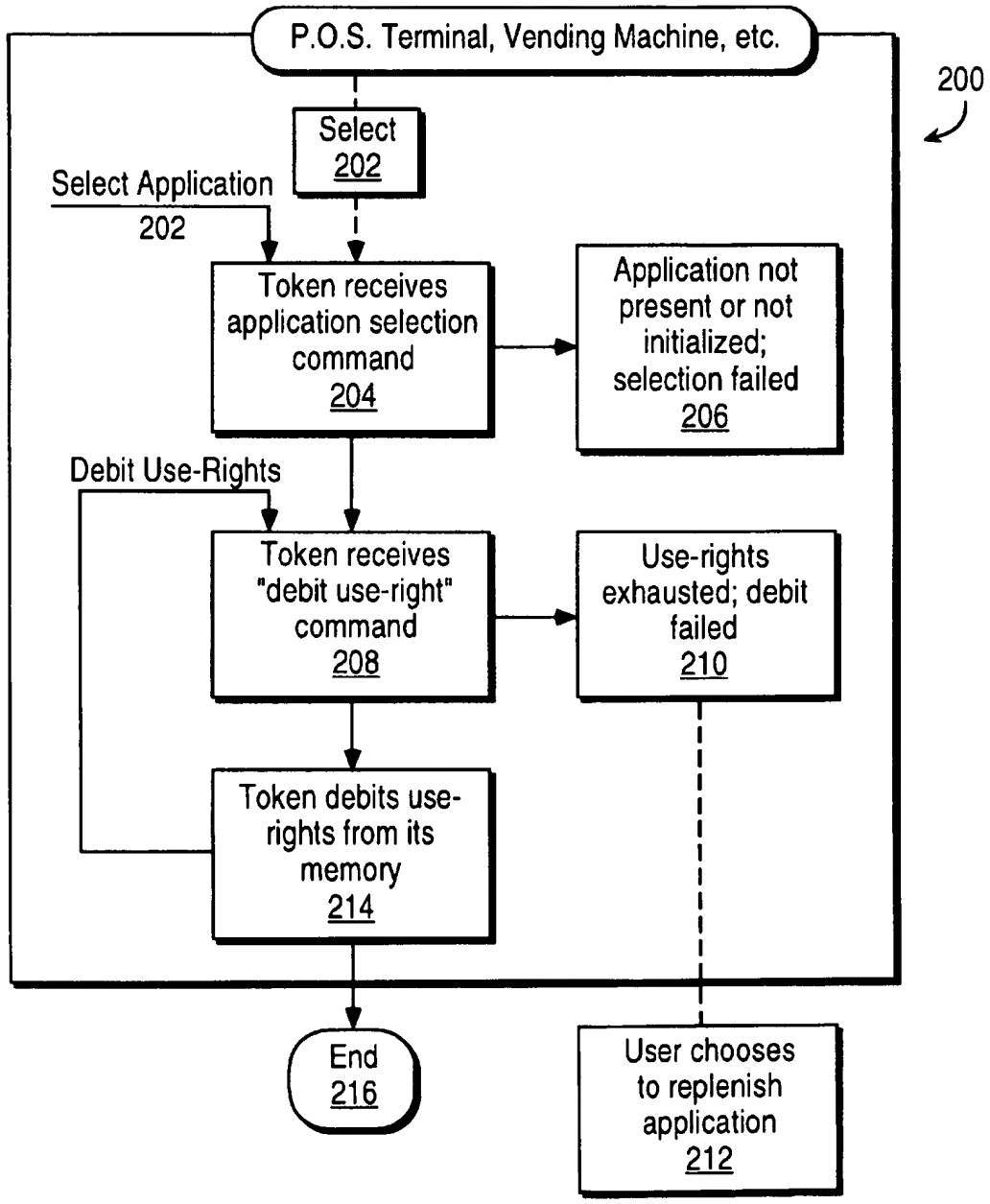
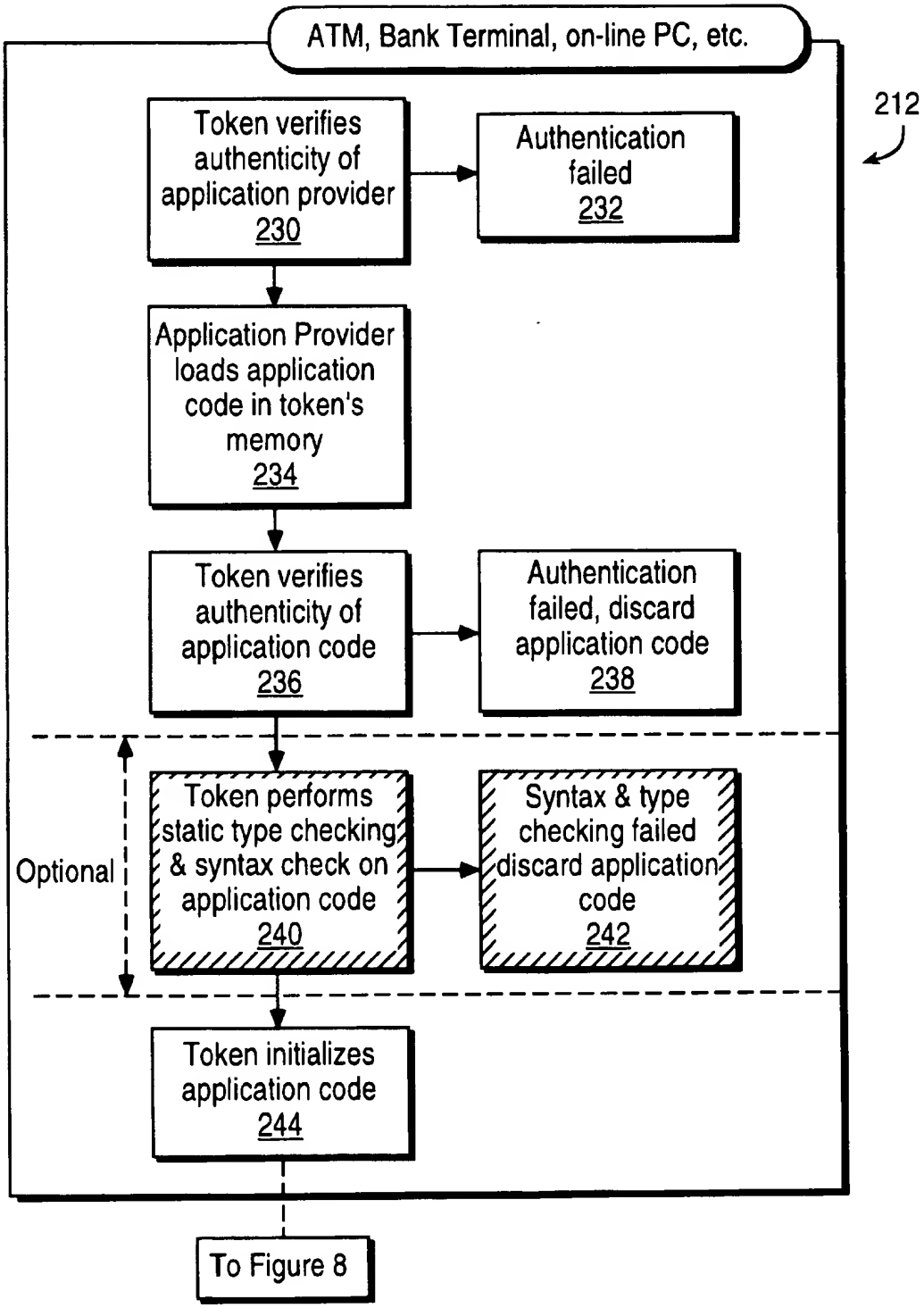

P.O.S. Terminal, Vending Machine, etc.
200
Select
202
Select Application
202
Token receives application selection command
204
Application not present or not initialized; selection failed
206
Debit Use-Rights
Token receives "debit use-right" command
208
Use-rights exhausted; debit failed
210
Token debits use-rights from its memory
214
End
216
User chooses to replenish application
212

FIG. 9


ATM, Bank Terminal, on-line PC, etc.
212
Token verifies authenticity of application provider 230
Authentication failed 232
Application Provider loads application code in token's memory 234
Token verifies authenticity of application code 236
Authentication failed, discard application code 238
Optional
Token performs static type checking & syntax check on application code 240
Syntax & type checking failed discard application code 242
Token initializes application code 244
To Figure 8

SYSTEM AND METHOD FOR LOADING APPLICATIONS ONTO A SMART CARD

BACKGROUND OF THE INVENTION

This invention relates generally to secure portable tokens, such as smart cards and in particular to smart cards having reloadable applications.

As is well known, a smart card may be a plastic, credit card-sized card containing a semiconductor chip, such as a microprocessor built into the smart card so that it may execute some simple application programs, which may be referred to as applets. Some examples of the applications in a smart card include security and authentication, information storage and retrieval, and credit and debit operations for managing value accounts, such as prepaid phone time and debit accounts. Each value account application on the smart card has a particular type of use rights associated with the application. For example, a prepaid phone time application may have a predetermined number of prepaid phone minutes that are used up as phone calls are made with the card, and a prepaid public transit account may have an initial preset monetary values which is debited with each use of public transportation. To store and execute these applets, these smart cards have a built-in memory and processor. In order to ensure the security of the use rights on these smart cards, only the processor within the smart card may ordinarily alter the value of the use rights, and only after an authorization sequence has been successfully conducted. The network in which the smart card is being used does not have any direct access to the memory of the smart card nor to the use rights of any application.

There are generally two different types of smart cards, i.e., disposable smart cards and permanent, non-disposable smart cards. A disposable smart card may have a rudimentary semiconductor chip embedded within the smart card and may have a limited amount of memory and some hardwired logic. The disposable smart cards may have a predetermined initial amount of prepaid use rights or other value stored in the memory of the smart card established when the smart card is manufactured. The prepaid use rights are then depleted as the smart card is used. A prepaid phone card or a subway fare card are examples of disposable smart cards because these smart cards are thrown away after the prepaid use rights are depleted. These disposable smart cards are inexpensive because of the rudimentary semiconductor chip, but they have limited utility since their stored value cannot be replenished, and other applications cannot be installed on them. Due to the limited memory and processing power, these disposable smart cards also cannot execute sophisticated cryptographic algorithms, which means that these disposable smart cards are less secure.

The non-disposable, permanent smart cards may have a more complex semiconductor chip embedded within the card, and may have a programmable micro-controller and an expanded memory. The memory may store one or more applets that have separate predetermined amounts of use rights for different functions. Importantly, these permanent smart cards have use rights that may be replenished so that the permanent smart card need not be discarded once the use rights are depleted. Examples of these permanent smart cards include banking cards according to the Europay/Mastercard/Visa standard, and pay television access control cards. These permanent smart cards have more memory for storage of multiple applets and the use rights on the smart card may be separately and independently replenished. However, these permanent smart cards are also more expensive due to the additional memory and the microcontroller, and the replenishment can only be performed by the card issuer.

Initially, many companies issued disposable smart cards due to the lower initial investment. However, due to the security concerns of these disposable smart cards and the limited applications that may be run on these disposable cards, the current trend is to use permanent smart cards because several applications may be loaded onto a single permanent smart card. The permanent smart card is also more secure because more sophisticated cryptographic techniques may be used.

Most conventional permanent smart cards may have a memory unit that may include a read only memory (ROM), a random access memory (RAM), and a non-volatile memory (NVM). The NVM may be, for example, a flash memory such as a flash electrically erasable programmable read only memory (Flash EEPROM), or a EEPROM. These permanent smart cards receive all of their power from the terminal to which they are connected during use. As a consequence, the RAM, which is volatile memory, may be used only as a scratch pad memory for simple computations that do not need to be stored. The ROM, which is permanent, may store the operating system (OS) of the smart card and other programs which do not need to be updated or changed, such as certain permanent applets. The NVM may store certain applets and the use rights secrets or values associated with all applications in the smart card. These conventional permanent smart cards may have multiple applications that reside in the memory of the smart card.

Some conventional permanent smart cards have fixed application programs that are stored in the ROM at the time that the smart card is manufactured. These smart cards do not permit any applications to be stored in the NVM due to security concerns. The programs that are stored in the ROM cannot be altered. The applications for these ROM-based smart cards, however, take a great amount of time to develop because the application must be developed and then be hard wired into the ROM. In addition, these fixed applications are not changeable or removable.

To solve the problems of a fixed application in the ROM, some current smart cards permit applications to be stored in the NVM. However, handling of applications and their associated use rights in the NVM of the smart card poses several problems.

First, there is a security problem since access to the application within the NVM may also permit access, by a clever individual, to the other applications within the NVM unless carefully controlled. In addition, a clever person may figure out a way to replenish his use rights illegally as they are also stored in the NVM. This is an especially large problem for banks that want to issue debit or electronic purse cards since a person could replenish the money available on the smart card without debiting his bank account. For a bank, it is desirable that no one, but the bank have access to the use rights within the smart card. This means that the use rights of any applet on a smart card may only be replenished by the card issuer, such as the bank, which may be inconvenient. In addition, any other company with applets on that smart card must have a relationship with the card issuer.

Second, the replenishing of the use rights of an applet in the smart card may be slow because there must be a number of security procedures that must be followed when use rights are being changed. For example, there must be several authentication procedures to ensure that no illegal activities are occurring.

Third, since each type of application may have a different type of use rights in various different units, such as phone minutes in time units versus cash in monetary units, each different application will probably require a different use rights reload procedure. For example, a use rights reload procedure for phone minutes may not be able to replenish the cash of a debit account on a smart card. Thus, procedures that loads use rights into the smart card must be duplicated.

To limit access to these use right values, conventional permanent smart cards have done several different things. First, some conventional permanent smart cards have controlled the access to certain areas of memory, known as memory zones, so that these memory zones are write-once areas. Other conventional permanent smart cards use a data dictionary, which keeps track of the memory areas in which each of the application must reside. Thus, some sort a memory management system must constantly verify that none of the applications are doing illegal activities.

In summary, some conventional permanent smart cards do not allow any applications to reside in the NVM to reduce security risks. Other conventional permanent smart cards have systems for replenishing the use rights of an application contained on a smart card, but limit this capability to the issuer of the smart card, and require separate loading procedures for each applet. None of these conventional smart card systems provide a system for loading an entire application of any type, including the use rights, into the memory of a permanent smart card. Accordingly, conventional smart cards cannot store disposable applications, such as a prepaid telephone time applet, because there is no method for removing the disposable application once it is depleted or replacing the disposable applet with a new applet. Thus, in conventional smart cards, these depleted disposable applications would remain in the smart card taking up valuable memory space. For this reason, most permanent smart cards today do not have any ability to handle disposable applications.

Thus, there is a need for a system and method for universally reloading different types of use rights in multiple application smart cards which avoid these and other problems of known devices, and it is to this end that the present invention is directed.

SUMMARY OF THE INVENTION

The invention provides a smart card, as well as a system and method for loading applications into the memory of a smart card which may load any type of application and its associated use rights, wherein the use rights may have any type of units. In addition, the system may load one or more disposable applications onto a permanent smart card since those disposable applications, once depleted, may be replaced with a new applet.

The invention also provides an applet loading system for a smart card wherein the use rights associated with an applet may be replenished by reloading the applet and the use rights into the memory of the smart card. The system for loading applications into a smart card may be universal so that a single loading system may be used for a variety of applications. In accordance with the invention, a system and method for reloading applications within a smart card is provided wherein the system may have a storage, remotely from said smart card, that stores an applet and use rights with a predetermined initial value, associated with the applet, and has a smart card having a processing unit, and a memory unit, the memory unit being connected to the processing unit and storing a second application having use rights. The smart card may be connected to said remote storage means, and the application, having use rights with a predetermined value, may be loaded from said remote storage means into said smart card. A smart card is also provided having a processor for executing an application, a memory, connected to the processor, for storing multiple applications, including a first application having first use rights and having first values associated with the first use rights, the first value changing from a predetermined initial value with use of the first use rights, a system for loading in the smart card a second application from a remote location over an interface, the second application having second use rights, a system for storing said second application into said memory in said smart card, and a system for changing the use rights of said first application and said second application. A method of replenishing the use rights in a smart card is also provided.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a smart card with which the invention may be employed;

FIG. **2** is a block diagram depicting the creation of a program that may run on the smart card of FIG. **1**;

FIG. **3** is a block diagram of the memory organization of the smart card of FIG. **1**;

FIG. **4** is a block diagram of a preferred system for reloading applications onto a smart card;

FIG. **5** is a block diagram of a first embodiment of a method in accordance with the invention of reloading an application into a smart card;

FIG. **6** is a block diagram of a second embodiment of a method in accordance with the invention of reloading an application into a smart card;

FIG. **7** is a block diagram of a third embodiment of a method in accordance with the invention of reloading an application into a smart card;

FIG. **8** is a flowchart of a method of debiting use rights in a smart card; and

FIG. **9** is a flowchart of a method of replenishing the use rights of an application within a smart card in accordance with the invention.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

The invention is particularly applicable to a system and method for reloading applications having use rights onto a permanent smart card so that the use rights of the application may be replenished when they have been depleted. It is in this context that the invention will be described. It will be appreciated, however, that the system and method in accordance with the invention has greater utility.

FIG. **1** is a block diagram of a smart card **20**, also known as a token, of the type with which the invention may be employed. The smart card may be used in connection with the system and method of loading applications into a smart card in accordance with the invention. The smart card may preferably be a permanent smart card, but may also be a disposable smart card. This smart card **20** may have a processor or CPU **22** and a memory **24**. The memory may comprise a read only memory (ROM) **26**, a random access memory (RAM) **28**, and a non-volatile memory (NVM) **30**. The NVM may be any type of writable nonvolatile memory, such as an electrically erasable, programmable read only memory (EEPROM), a battery backed RAM, or a flash memory, that can retain stored data when no electrical power is supplied to the memory. The ROM may preferably store

the operating system (OS) which controls the operation of the CPU of the smart card, and the RAM may be used as a temporary scratchpad memory. Because the smart card receives its electrical power from the terminal into which it is inserted, as described below, all of the contents of the RAM will be lost when the smart card is removed from the terminal. The NVM may preferably be used to store one or more applications which may be referred to as applets due to the small size of the actual program code. Each of these applets may have associated use rights which are specific to the applet. Other permanent applications that do not change, such as a credit/debit program, may be stored in the ROM.

The processor **22** controls the operation of the smart card. The processor may be connected to all of the memories within the memory system **24**. Since there are use rights associated with an application, there is a need to make the smart card secure to prevent theft or alteration of the use rights. To accomplish this security, the processor is the only system that is capable of accessing any of the memories. There is no direct access to any of the memories from outside of the smart card. In addition, any outside access to the memories of the smart card must be conducted through an input/output (I/O) line **32** that is connected to the processor **22**. The smart card may also have more than one I/O line provided that access to each I/O line is carefully controlled so that there is no direct access to any of the memories from outside of the smart card. Thus, the processor may authenticate and validate incoming requests prior to making any change in the use rights of an application stored in the smart card, and may prevent unwanted or illegal attempts to decrease the use rights of an application. This authentication and validation may be conducted using cryptographic systems, such as public key encryption, or any other security system. Now, a preferred system for generating applets for a smart card will be briefly described.

FIG. **2** is a block diagram showing the architecture of the smart card and the manner in which an applet is generated for the smart card. To provide sufficient security for the smart card, a preferred embodiment of a smart card may have a virtual machine **40** contained within the smart card. The virtual machine is comprised of a software interpreter **42** running on the hardware processor **22**. The interpreter is a piece of software that acts as an interface between the hardware processor and the applets. In this manner, the applets run through the interpreter so that the applets do not have any direct access to the hardware of the smart card. Thus, the interpreter may verify that none of the applets are performing illegal operations. Instead of a complete interpreter and virtual machine, the smart card may have a command dispatcher to control the access of the applets to various portions of the smart card. The dispatcher may control access of the applets to the hardware by preventing the applets from receiving any access until an authentication check has been completed. A command dispatcher may be considered to be a reduced version of a general interpreter, and the command dispatcher interprets commands received from the applications instead of interpreting the entirety of the code of the applications.

To execute an applet on an interpreter, as shown, source code **46** of an applet is compiled into a byte code **48**. The byte code may then be executed by any interpreter on any smart card. The details of the architecture of the preferred smart card are set forth in more detail in PCT Application No. PCT/NL95/00055, published as International Publication No. WO 95/22126, which is incorporated herein by reference. The organization of programs within the memory of the smart card will now be described.

FIG. **3** is a block diagram of the memory organization of the smart card **20** that may include a system for loading applets into the smart card in accordance with the invention. The memory **24** of the smart card, which may include the ROM and NVM, may be logically organized into an OS layer **50**, an executive layer **52**, and an application layer **54**. The OS layer may contain the most basic operating software, such as a cryptographic library **56**, and an interpreter **58**. These programs are permanent and may be stored in the ROM. The cryptographic library may be used for authenticating access to the smart card, as described above. The interpreter **58**, as described above, may be used to prevent an applet from directly accessing the hardware of the smart card.

The executive layer **52** may contain, for example, an application launcher **60**, a conditional application loader **62** in accordance with the invention, and other OS sub-systems **64**. The application launcher receives a request to access an application, and after appropriate authentication, launches and controls the applet. The conditional application loader **62** controls the loading of an application, or applet, into the NVM of the smart card. The application loader may verify that the remote system desiring to load an applet into the smart card has the appropriate authority, and then may perform the necessary operations, as described in more detail below, to load the applet into the NVM of the smart card.

The application layer **54** may contain a permanent application **66** and one or more disposable applications **68** having associated use rights. The permanent application may be stored in the ROM since it is permanent and may be a credit/debit system that performs all of credit and debit transactions for all of the disposable applications having use rights within the smart card. The credit/debit system may operate with any type of use rights so that only a single credit/debit application is needed for each smart card. In this manner, the use rights of any applet within the smart card may be changed by the permanent credit/debit application **66**. In a preferred embodiment of the invention, the loader **62** and the credit/debit application **66** may be a single program since both programs operate on all of the applets having use rights. For example, an applet with use rights needs the credit/debit application to authorize the reload if the applet when the use rights have been depleted, as described below.

The disposable application **68** may be any type of application or applet with a limited lifetime, as defined by a certain number of use rights, such as a predetermined number of telephone call minutes, a predetermined amount of money, or a predetermined number of store credits. As described below in more detail, conventional smart cards that replenished the use rights of a particular application require a separate use rights loading system for each different application because the use rights of each application may require different handling and security. For example, replenishing a certain number of store frequent buyer points onto a smart card may be different than replenishing the cash value of a debit applet, such as a point-of-sale applet, in the smart card. In addition, in order to replenish the use rights of any applet, the smart card needed to be physically connected with or returned to the card issuer since only the card issuer had the authority the alter the use rights for an applet. Therefore, every company who may have an applet on the smart card, must have a relationship with the card issuer so that the card issuer can replenish the use rights of that applet.

Significantly, however, the smart card in accordance with the invention may have a universal applet loader that may

delete and then reload an entire applet instead of establishing a connection between the smart card and the applet issuer who then just reloads the use rights. Reloading the entire applet into the smart card means that the loader does not have to be specialized to handle the multiplicity of different types of use rights which could be present in the smart card since the entire applet, including the use rights, is being reloaded into the smart card. The loading of an applet into a smart card to permit the replenishment of the use rights of an applet will be described in more detail below.

The universal loader **62** in accordance with the invention may also be used to load new applets into a smart card, provided that the smart card has available memory. In addition, the universal loader may also permit an applet with depleted use rights to be deleted from the memory of the smart card and replaced with a new different application having refreshed use rights. Each of these operations will be described in more detail below. A preferred system, external to the smart card, for loading applets having use rights, into the smart card will now be described.

FIG. **4** is a block diagram showing a system in accordance with the invention for loading an applet having use rights into a smart card. The system may include the smart card **20**, a terminal **80**, and a server **82**. The smart card **20** is described above with reference to FIGS. **1–3**. The terminal may be operated by the smart card issuer, or by some other entity, such as a bank. The terminal may be a bank ATM teller, a terminal in a bank or a home computer system. The server may be maintained by a bank or the issuer of the smart card, and may contain downloadable applets. The connection between the terminal and the server may be any conventional network, such as the usual connection between ATM machines across the world.

As described above, the smart card may have the processor **22**, the OS layer **50** and the executive layer **52** stored in the ROM **26**, and the applications layer **54** stored in the NVM **30**. In addition, the smart card may have an interface system **86** that may connect the smart card to the terminal **80** using a corresponding interface **88**. A second interface **90** may connect the terminal to the server **82** via an interface **92**. Thus, the smart card may be connected, through the terminal, to the server. A preferred method of loading an application into the smart card will now be described.

When the smart card is connected to the terminal, the processor **22**, using the loader **62**, verifies the authenticity of the terminal and of the server. The terminal and the server may also verify the authenticity of the smart card. For example, when the smart card is connected to the terminal, the user may enter a personal identification number (PIN) that may be verified by the server. As another example, the server may send a coded word that must be correctly answered by the smart card. If the server and the smart card authenticate each other, then the universal loader **62** within the smart card begins the loading process. The applets stored on the server, regardless of the type of use rights, may all have a common structure so that the universal loader does not have to distinguish between different types of applets except to identify which one(s) to load. As shown, the NVM **30** may currently store the permanent credit/debit application **66**, and an existing first applet **94** with use rights. After the loading operation, as described below, the NVM memory may also have a second new applet **96** with use rights. In the smart card shown, the use rights of the first applet **94** have been depleted. Therefore, a new copy of the applet **98** with refreshed use rights, located on the server **82**, may be loaded into the NVM of the smart card. The applet **98** with refreshed use rights replaces the original applet **94** with depleted use rights.

In addition to the replenishment of use rights, a new **100** applet having use rights may be loaded into the smart card **20** from the server **82** in a similar manner. Therefore, after the load process is complete, the smart card may have a first applet with replenished use rights, and the new second applet **96** with predetermined use rights. As an example, a smart card that has a telephone call applet with depleted use rights may have a new telephone call applet with refreshed use rights as well as a debit applet with a predetermined value, e.g., $100, loaded onto the smart card. The connections between the terminal **80** and the server **82** may be conventional network system that may be used for home banking and the like. Several examples of loading applets into a smart card, in accordance with the invention, will now be described.

As described above, conventional smart cards replenish the use rights of an applet by reloading new use rights into an applet on the smart card. The problems with reloading the use rights of an applet into a smart card have been described above. Now, several examples of the operation of the applet loading system in accordance with the invention will be described.

FIG. **5** is a block diagram of the loading system in accordance with the invention being used to replenish the use rights of an applet within a smart card. As shown, the smart card **20** may have, for example, a first applet **102**, a second applet **104**, and a third applet **106**. In this example, the first and third applets have use rights remaining, whereas the second applet needs to have its use rights replenished. In accordance with the invention, a new second applet **108** with replenished use rights is loaded into the smart card **20** and replaces the old second applet **104**. Thus, after the loading process, the smart card may have a first applet **102**, a third applet **106**, and a new second applet **108** with replenished use rights. As shown, only the second applet is affected by the loading process. As described above, since the entire applet is loaded back into the smart card, the type of the use right of the applet is irrelevant, and the loading system may reload any type of applet within the smart card regardless of the type of use rights that the applet may have.

FIG. **6** is a block diagram of the loading system in accordance with the invention being used to load a disposable application onto an existing smart card. As shown, the smart card **20** may have a first applet **102**. In addition, at a remote system **112**, a disposable applet **114** may be stored. The disposable applet may be loaded into the smart card **20** so that the smart card may contain the first applet **102** and the new disposable applet **114**. The disposable applets may be easily loaded into the smart card. In addition, once the use rights of the disposable applet are exhausted, the disposable applet may be replaced, using the loading method in accordance with the invention, with a new applet having new use rights.

For example, a user may take a trip to a foreign country and desire some local currency to be placed on the smart card so that he does not have to carry any cash. At the end of the trip, the user does not want to keep the foreign currency applet since he will not have any further need for it. Thus, the invention enables the foreign currency applet to be replaced by, for example, a prepaid telephone call applet.

FIG. **7** is a block diagram of the loading system in accordance with the invention being used to replenish the use rights of an applet in a smart card. In this example, the smart card **20** has a single applet **116** with use rights. After some time, the use rights of the applet have been depleted. In accordance with the invention, the applet **116** may be

replaced by a new applet **120** that has the same functions as the old applet, but has replenished use rights.

The invention, as shown, is not limited to any particular number of applets and may by used to replenish the use rights of as few as a single applet or to replenish the use rights multiple applets. The invention may also be used to load and replace a single disposable applet onto a smart card. A method of debiting use rights in a smart card will now be described.

FIG. **8** is flowchart of a method **200** of debiting use rights in a smart card. First in step **202**, an applet within the smart card may be selected. For example, when a smart card is placed into a telephone terminal, then the applet with the telephone use rights may be selected by the terminal. In order to select the applet, the smart card may verify that the terminal has the proper authority to access that particular applet. Then, at step **204**, the smart card receives an application selection command from the terminal, for example. If the application selected is not initialized or present in the smart card, the method ends in step **206**. If a valid application is selected, then in step **208**, after a debit use rights command is issued, the smart card receives a debit use rights command at step **208**. If the use rights have been exhausted already, then in step **210**, the debit fails, and in step **212**, the use rights of the applet may be replenished, as described below. If a valid debit command is received, then in step **214**, the decreased use rights of the applet are calculated and stored in the memory of the smart card. Then, if there are additional debits for the applet, the method loops to step **208**, otherwise the method ends at step **216**. The method of replenishing the use rights for an applet on the smart card in accordance with the invention will now be described.

FIG. **9** is a flowchart of the step **212** of FIG. **8**, for replenishing the use rights of the applet in accordance with the invention. The applet may be selected because it has expended its use rights or because the user selects a particular applet. As described above, the universal loader can load any type of applet with any type of use rights from the server to the memory of the smart card. In addition, since the loader can load any type of applet, it is not necessary to get the use rights of the applet reloaded by the card issuer. Thus, the universal loader permits a greater amount of flexibility.

Once any of the applet with the associated use rights has been selected, at step **230**, the smart card verifies the authenticity of the provider, such as the server, of the applet. If the authentication fails, then the method ends at step **232**. If the authentication is successful, then in step **234**, the provider, with the help of the loader, loads the applet into the NVM of the smart card.

Typically, authentication of the applet code may be achieved by the smart card through the verification of a digital signature, a cryptographic check sum or a predetermined hash value. In step **236**, the smart card verifies the authenticity of the program code of the applet to detect viruses, and the like. In step **238**, if the authentication of the applet code fails, then the applet code is deleted from the memory of the smart card.

The next step is an optional step that is not required in order to load an application into a smart card in accordance with the invention. This step requires a smart card with a larger amount of memory. In this optional step **240**, the smart card may perform static type checking and a syntax check of the code of the applet. If this check fails, then in step **242**, the applet code is deleted from the memory of the smart card. In the last step **244**, the smart card initializes the code of the applet so that the use rights of the applet may be debited, as described above with reference to FIG. **8**.

While the foregoing has been with reference to a particular embodiment of the invention, it will be appreciated by those skilled in the art that changes in this embodiment may be made without departing from the principles and spirit of the invention, the scope of which is defined by the appended claims.

We claim:

1. A system for loading an application and its associated use rights into a smart card having other applications, some of the other applications with associated use rights that have values that change as the application is used, the system comprising:

means for storing, remotely from said smart card, an application and use rights with a predetermined initial value, associated with the application;

said smart card having a processing unit, a memory unit for receiving applications having different use rights, the memory unit being connected to the processing unit and storing a second application having associated use rights;

means for connecting said smart card to said remote storage means; and

means for loading said application, having use rights with a predetermined value, from said remote storage means into said smart card.

2. The system of claim **1**, wherein the use rights have a refreshed state and a depleted state, the use rights of the second application being depleted and the use rights of the application being refreshed, and further comprising means for replacing said second application stored in the memory with said application at the remote storage means so that the use rights of the application in the memory are replenished.

3. The system of claim **2**, wherein the connecting means further comprises means for verifying the authority of the remote storage means to load an application into the memory of the smart card.

4. Smart card apparatus for loading an application having use rights with values which meter use of the application, the smart card comprising:

a processor for executing an application;

a memory, connected to the processor, for storing multiple applications, including a first application having first use rights and having first values associated with the first use rights, the first value changing from a predetermined initial value with use of the first use rights;

an interface enabling the processor of said smart card to communicate with a remote location;

means for receiving, in the smart card, applications having different use rights, the applications including the first application and a second application from said remote location over said interface, the second application having second use rights; and

means for storing said second application into said memory in said smart card.

5. The smart card apparatus of claim **4** further comprising means for replacing said first application stored in the memory with said second application from said remote location so that the use rights of the application in the memory are replenished.

6. The smart card apparatus of claim **5**, wherein the receiving means further comprises means for verifying the authority of the remote location to load an application into the memory of the smart card.

7. A method of replenishing use rights in an application stored in a smart card, the use rights having a refreshed state and a depleted state and being depleted with use of the

application, the smart card having a processor and a memory for storing the application, the method comprising:

connecting a smart card having a first application with use rights in a depleted state to a communications system, the communications system being connected to a system remotely located from said smart card, the system storing a second application having equivalent use rights to the first use rights, the equivalent use rights having a refreshed state;

verifying in the card that said remote storage system has the authority to replace the first application in the smart card; and

replacing the first application in said memory with said second application having refreshed use rights so that the use rights of the application located within the memory of the smart card are replenished.

8. The method of claim **7**, wherein replacing further comprises deleting said first application from said memory of said smart card, and loading said second application having refreshed use rights from said remote storage location into said memory of said smart card so that the use rights of the application located within the memory of the smart card are replenished.

9. A method of loading an application into a smart card, the application having use rights with a refreshed state and a depleted state and being depleted with use of the application, the smart card having a processor and a memory for storing the application, the method comprising:

connecting a smart card having a first application with use rights to a communications system, the communications system being connected to a system remotely located from said smart card, the system storing a second application having use rights;

verifying in the smart card that said remote storage system has the authority to load the second application into the smart card; and

loading said second application having refreshed use rights into the memory of the smart card so that the second application may be used.

10. The method of claim **9**, wherein the first application has depleted use rights, the second application having refreshed equivalent use rights to the first application, and wherein the loading comprises replacing the first application in said memory with said second application having refreshed use rights so that the use rights of the application located within the memory of the smart card are replenished.

11. Smart card apparatus for loading an application having use rights with values which meter use of the application, the smart card comprising:

a processor for executing an application;

a memory, connected to the processor, for storing multiple applications, including a first application having first use rights and having first values associated with the first use rights, the first value changing from a predetermined initial value with use of the first use rights;

means for loading, into the smart card, applications having different use rights, the applications including a first application and a second application from a remote location over an interface, the second application having second use rights;

means for storing said second application into said memory in said smart card; and

means for changing the use rights of said first application and said second application.

12. The smart card apparatus of claim **11**, where said second application has equivalent use rights to the first use rights, the equivalent use rights having a refreshed state, and wherein storing means further comprises means for replacing the first application in said memory with said second application having refreshed use rights so that the use rights of the application located within the memory of the smart card are replenished.

* * * * *

US005715431A

United States Patent [19]

Everett et al.

[11] Patent Number: 5,715,431

[45] Date of Patent: Feb. 3, 1998

[54] **TAMPER PROOF SECURITY MEASURE IN DATA WRITING TO NON-VOLATILE MEMORY**

[75] Inventors: **David B. Everett**, East Sussex; **Keith M. Jackson**, West Sussex; **Ian Miller**, Surrey, all of United Kingdom

[73] Assignee: **Mondex International Limited**, London, United Kingdom

[21] Appl. No.: **351,451**

[22] PCT Filed: **Apr. 13, 1994**

[86] PCT No.: **PCT/GB94/00775**

§ 371 Date: **Feb. 3, 1995**

§ 102(e) Date: **Feb. 3, 1995**

[87] PCT Pub. No.: **WO94/24673**

PCT Pub. Date: **Oct. 27, 1994**

[30] **Foreign Application Priority Data**

Apr. 13, 1993 [GB] United Kingdom 9307623

[51] **Int. Cl.**[6] **G06F 12/14**

[52] **U.S. Cl.** **395/483**; 235/492; 902/26

[58] **Field of Search** 395/430, 492, 395/483, 486, 182.04, 182.05, 182.13, 186, 217; 235/380; 365/185.33; 902/26; 257/922

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,827,115	5/1989	Uchida et al.	235/492
5,200,600	4/1993	Shinagawa	235/492
5,365,045	11/1994	Iijima	235/380
5,386,539	1/1995	Nishi	395/430
5,390,148	2/1995	Saito	365/189.01
5,475,697	12/1995	Katz et al.	395/486
5,479,637	12/1995	Lisimaque et al.	395/430

FOREIGN PATENT DOCUMENTS

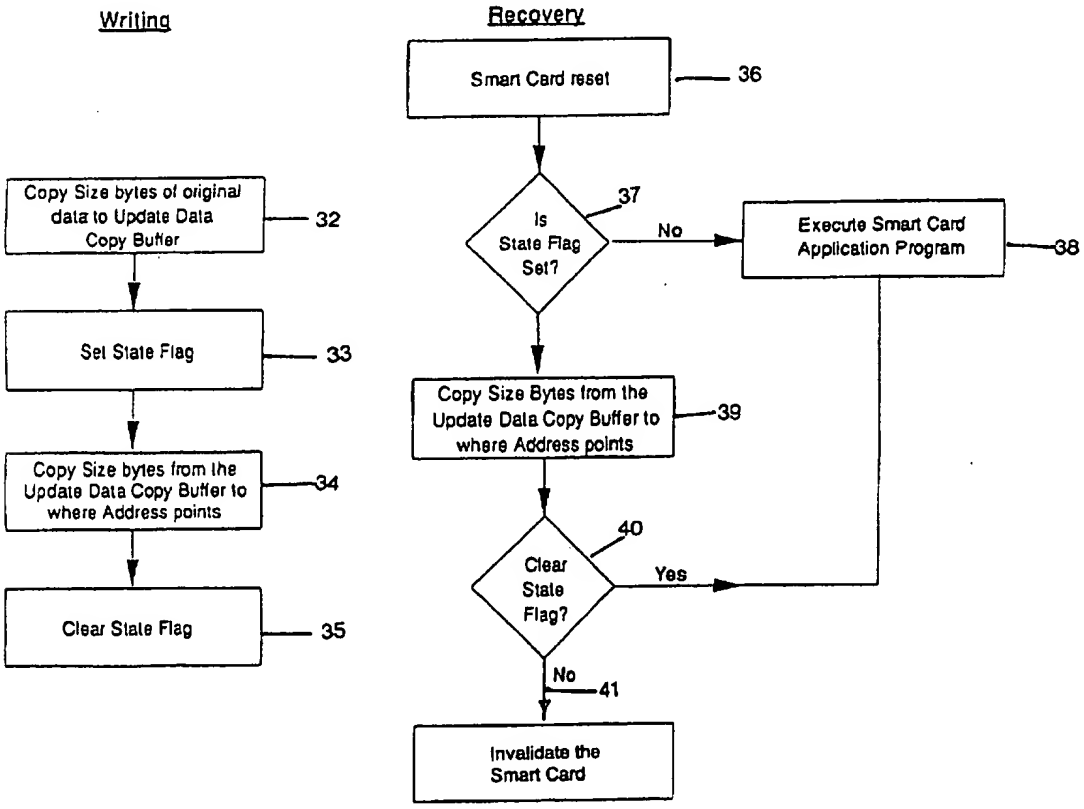
0 489 204 A1	6/1992	European Pat. Off. .

Primary Examiner—Eddie P. Chan
Assistant Examiner—Hiep T. Nguyen
Attorney, Agent, or Firm—Nixon & Vanderhye P.C.

[57] **ABSTRACT**

A method of writing data to non-volatile memory such as electrically erasable programmable read only memory (EEPROM) in a smart card provides a write status region of EEPROM which is examined on each reset of the card. If the preceding write operation was unsuccessful, perhaps because of deliberate manipulation of the card, a recovery procedure is implemented. If recovery is successful, the card operation can be run. Otherwise the card is unusable.

15 Claims, 6 Drawing Sheets



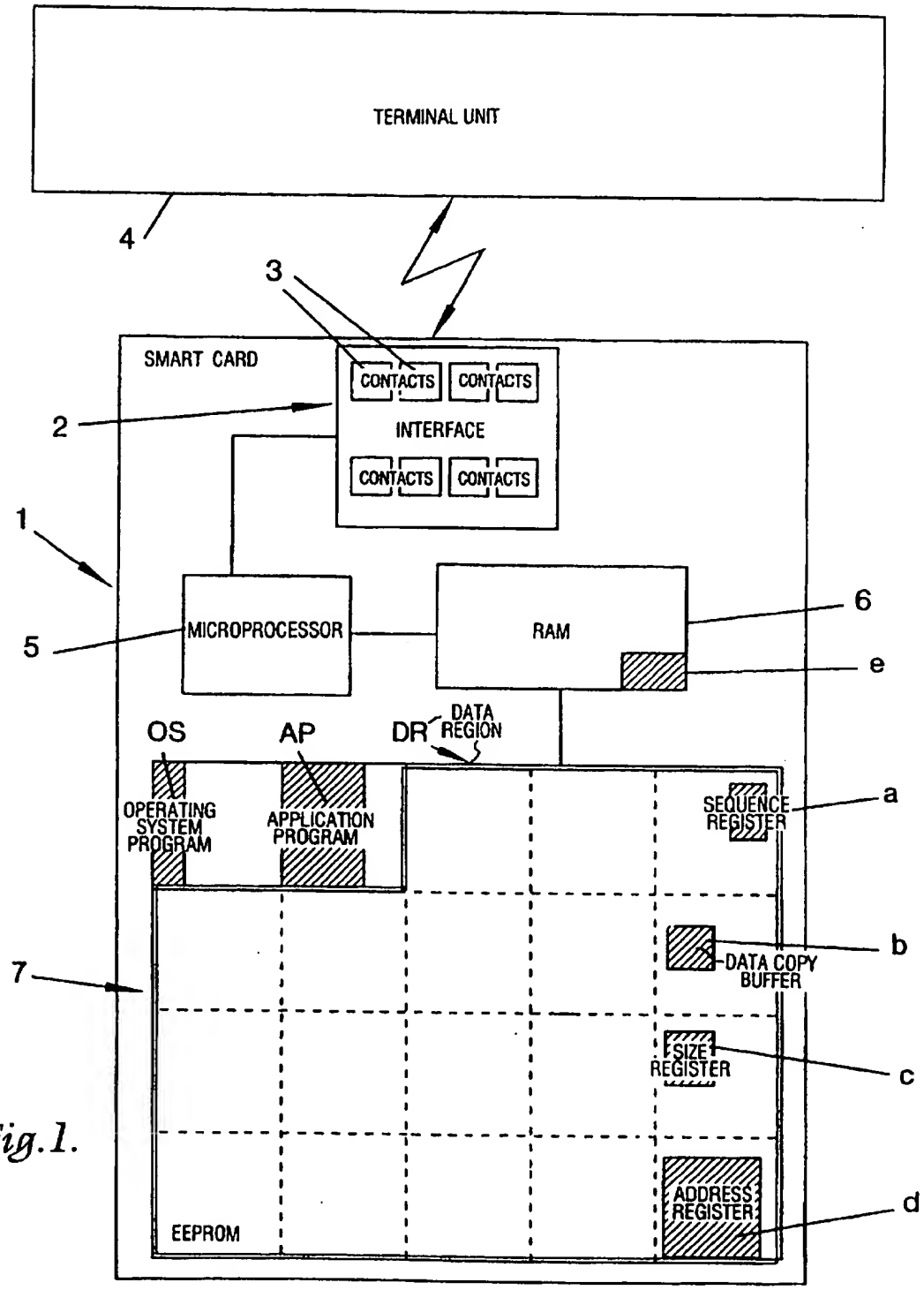

TERMINAL UNIT
4
3
SMART CARD
CONTACTS
CONTACTS
2
INTERFACE
CONTACTS
CONTACTS
1
MICROPROCESSOR
RAM
6
5
e
OS
AP
DR
DATA REGION
OPERATING SYSTEM PROGRAM
APPLICATION PROGRAM
SEQUENCE REGISTER
a
b
DATA COPY BUFFER
7
SIZE REGISTER
c
ADDRESS REGISTER
d
EEPROM

Fig. 1.

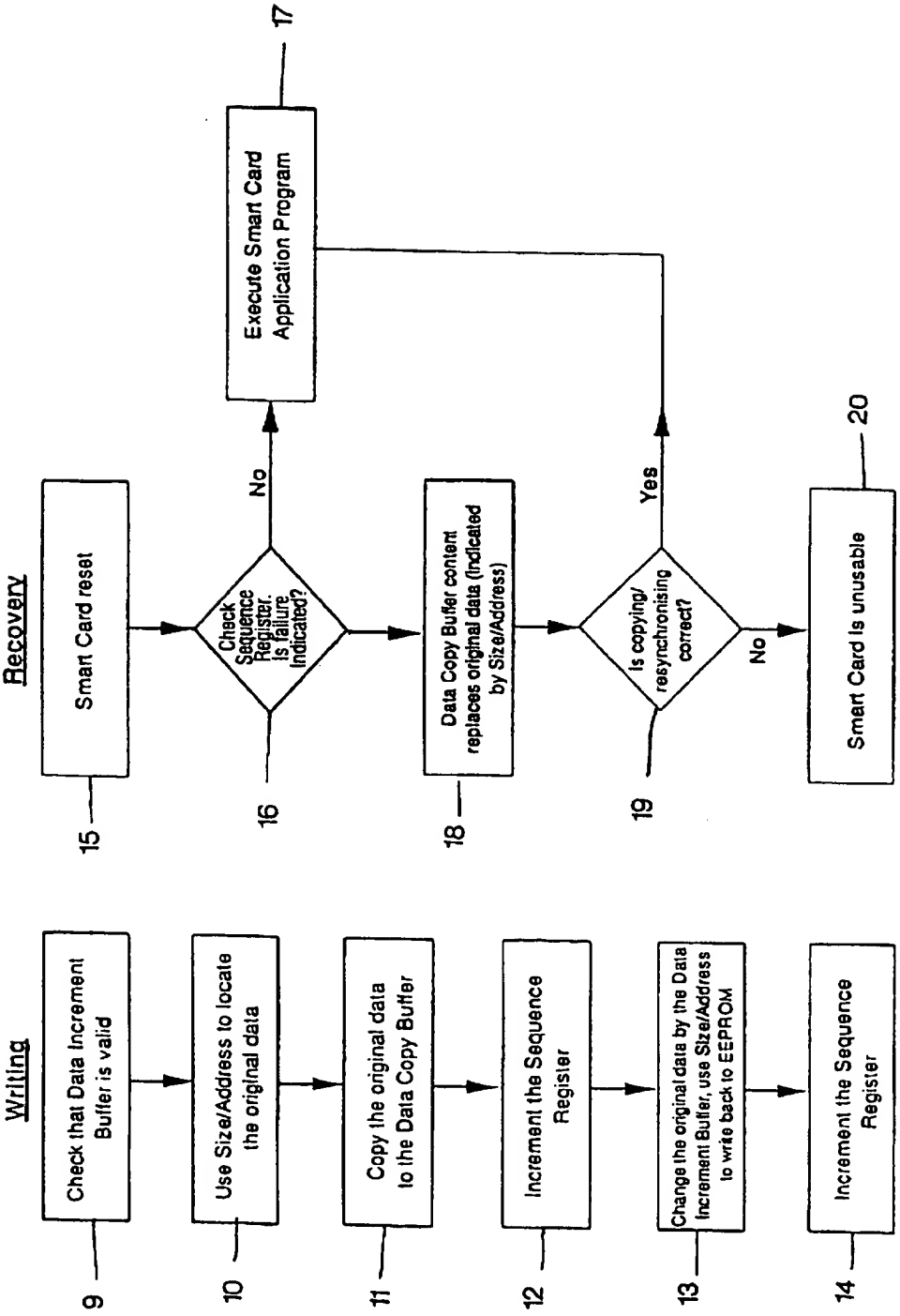

Writing
Check that Data Increment Buffer is valid
Use Size/Address to locate the original data
Copy the original data to the Data Copy Buffer
Increment the Sequence Register
Change the original data by the Data Increment Buffer, use Size/Address to write back to EEPROM
Increment the Sequence Register
9
10
11
12
13
14

Fig.2a.

Recovery
Smart Card reset
Check Sequence Register. Is failure indicated?
No
Execute Smart Card Application Program
Data Copy Buffer content replaces original data (indicated by Size/Address)
Is copying/ resynchronising correct?
Yes
No
Smart Card is unusable
15
16
17
18
19
20

Fig.2b.

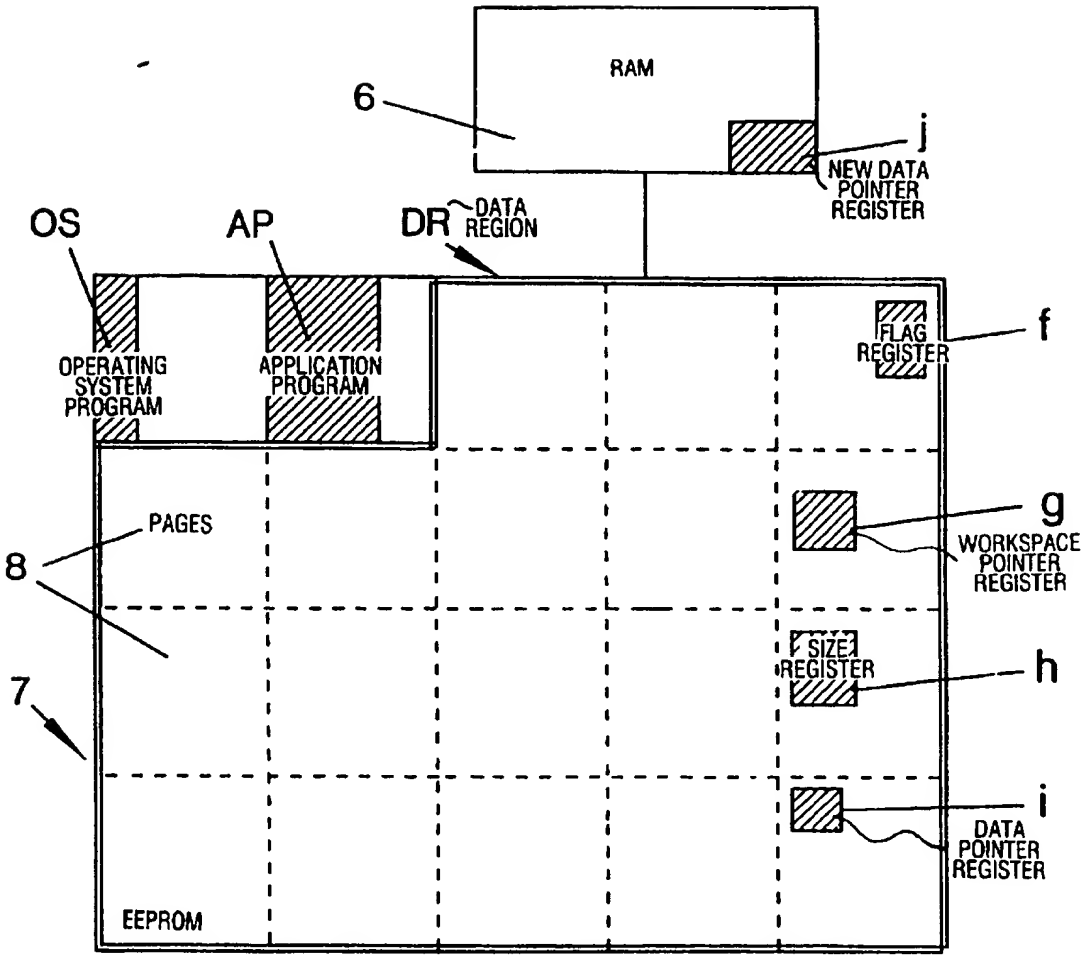

RAM
6
j
NEW DATA POINTER REGISTER
OS
AP
DR
DATA REGION
OPERATING SYSTEM PROGRAM
APPLICATION PROGRAM
FLAG REGISTER
f
PAGES
8
g
WORKSPACE POINTER REGISTER
SIZE REGISTER
h
7
i
DATA POINTER REGISTER
EEPROM

Fig.3.

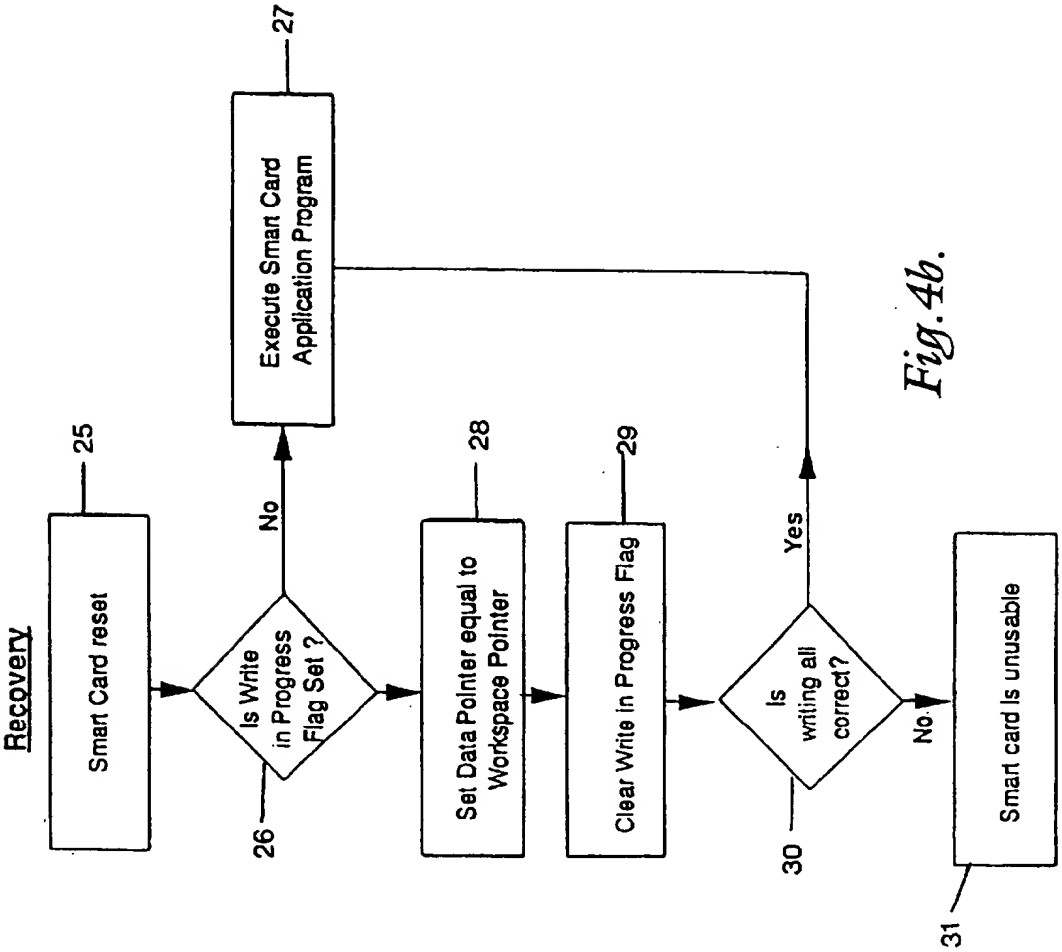

Recovery
Smart Card reset
25
Is Write in Progress Flag Set ?
26
No
Execute Smart Card Application Program
27
Yes
Set Data Pointer equal to Workspace Pointer
28
Clear Write in Progress Flag
29
Is writing all correct?
30
No.
Smart card is unusable
31

Fig. 4b.

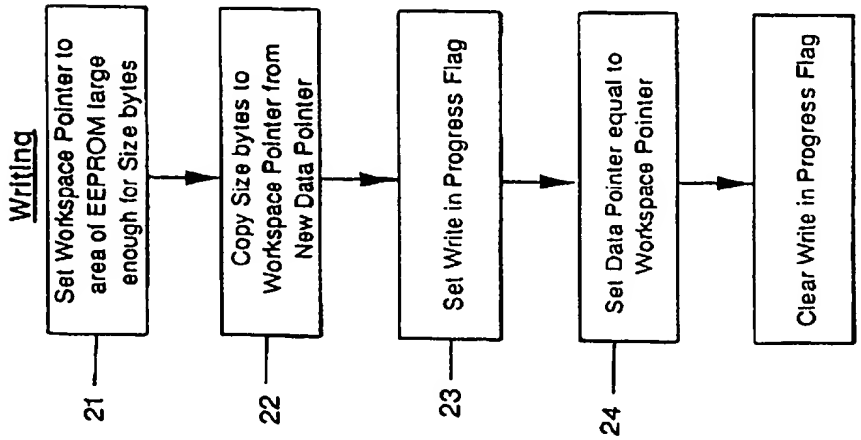

Writing
Set Workspace Pointer to area of EEPROM large enough for Size bytes
21
Copy Size bytes to Workspace Pointer from New Data Pointer
22
Set Write in Progress Flag
23
Set Data Pointer equal to Workspace Pointer
24
Clear Write in Progress Flag

Fig. 4a.

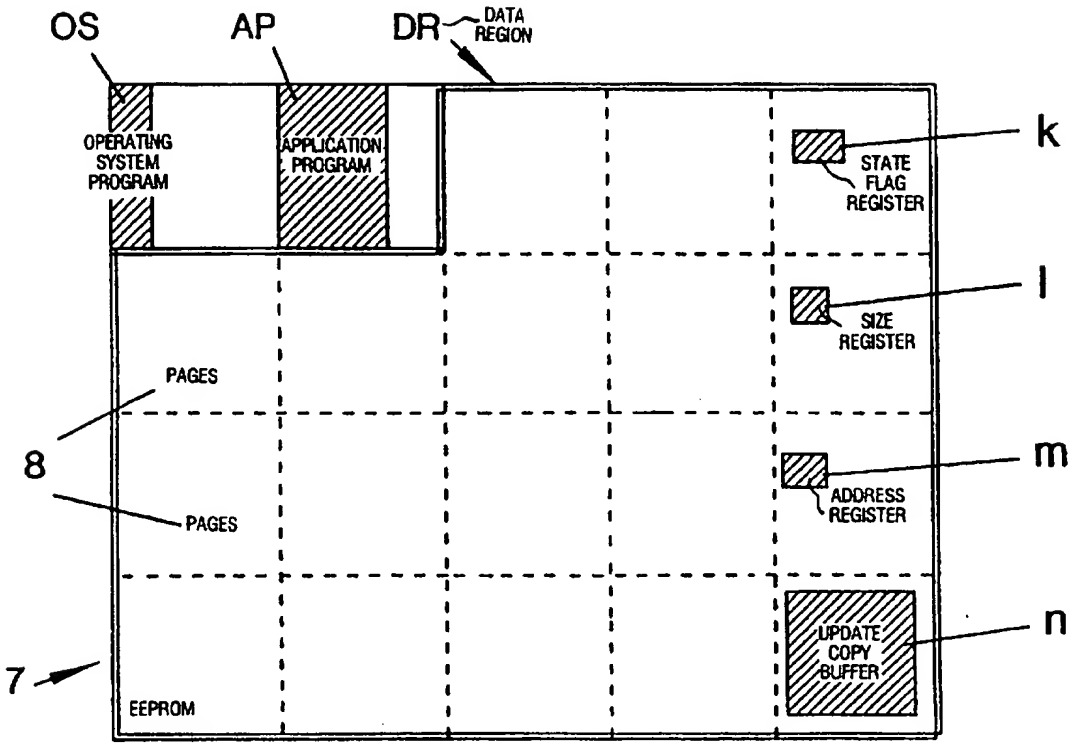

OS
AP
DR
DATA REGION
OPERATING SYSTEM PROGRAM
APPLICATION PROGRAM
STATE FLAG REGISTER
k
SIZE REGISTER
l
PAGES
8
PAGES
ADDRESS REGISTER
m
UPDATE COPY BUFFER
n
7
EEPROM

Fig.5.

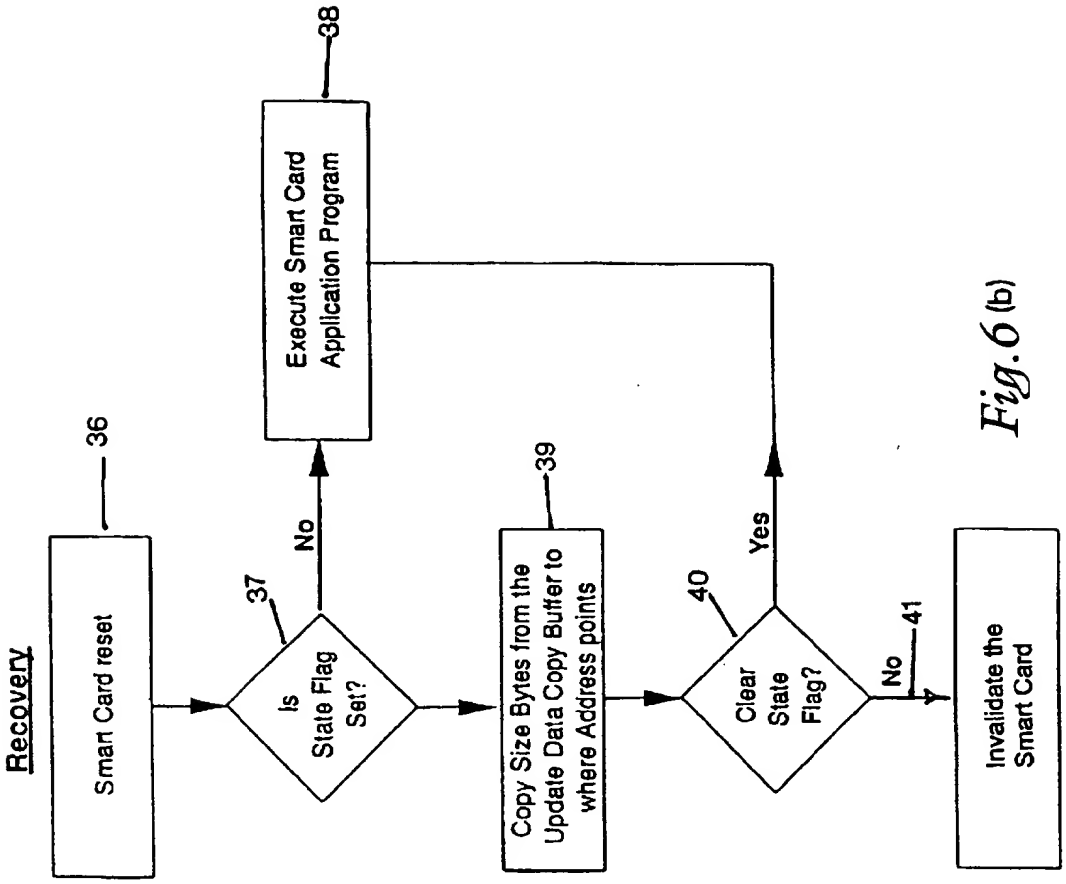

Recovery
Smart Card reset
36
Is State Flag Set?
37
No
Execute Smart Card Application Program
38
Copy Size Bytes from the Update Data Copy Buffer to where Address points
39
Clear State Flag?
40
Yes
No
41
Invalidate the Smart Card

Fig. 6 (b)

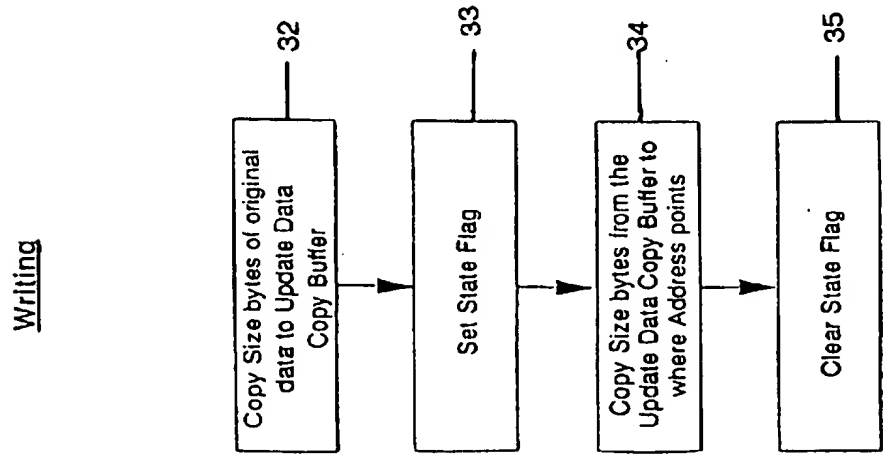

Writing
Copy Size bytes of original data to Update Data Copy Buffer
32
Set State Flag
33
Copy Size bytes from the Update Data Copy Buffer to where Address points
34
Clear State Flag
35

Fig. 6. (a)

TAMPER PROOF SECURITY MEASURE IN DATA WRITING TO NON-VOLATILE MEMORY

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to the writing of data to non-volatile memory. Non-volatile memory is memory which retains data without electrical power being maintained. In particular, the invention relates to the writing of data to memory in transportable integrated circuit devices which are used in conjunction with terminal devices with which they are temporarily coupled for data input and output. An example of such a transportable device is the integrated circuit card (ICC), otherwise known as a "smart card".

2. Discussion of Prior Art

Smart cards are coupled by means of an interface to a terminal device whereby power, clock signals, a reset signal and serial data signals may be applied to the card. Generally the interface incorporates a set of electrical contacts for direct temporary electrical connection. However, contactless interfaces employing electromagnetic induction techniques for the application of power have been proposed. In such an arrangement clock, reset and data signals may be coupled electromagnetically or by infra-red or ultra-sonic techniques. Transportable integrated circuit devices may be embodied in tokens of other than card shape. Regardless of shape, such devices will be referred to herein as integrated circuit cards (ICCs). A difficulty with ICCs is that the writing of data to the ICC may be interfered with by disturbing the interface during writing whereby transients or failure in power, reset or clock signals may result in erroneous write.

A smart card application to which the invention is particularly applicable is in a financial value or "electronic cash" transfer system. Here, data in smart cards represents value which can be transferred on-line with banks and off-line between cards. Such a system is described in patent applications Nos. WO91/16691 and WO93/08545. It is clearly important in such applications to avoid the effects of erroneous data writing, either accidental or perhaps deliberately instigated by manipulation of power or data lines. The present invention provides a solution.

SUMMARY OF THE INVENTION

According to the invention there is provided a method of writing data to non-volatile memory in an integrated circuit device, the device having an interface for temporary connection to a terminal unit; a microprocessor; random access memory and non-volatile memory, the method consisting in allocating a first region of the non-volatile memory for data to be written, allocating a second region of non-volatile memory for write status information to be written, performing a data write operation to write data to said first region, and writing information to said second region signifying a valid data write if, and only if, the data write operation is performed satisfactorily.

In a microprocessor environment there are many copy and write procedures for transferring data and program information between regions of RAM and from RAM to EEPROM, for example, and vice versa. At the operating system level or higher there are usually verification techniques available for verifying the validity of a copy or write operation. This may involve an automatic comparison of the copied or written material with the original or, more usually, the provision of a checksum routine which adds one or more checksum bits to the data which, in accordance with a particular algorithm, provide a link to the data which can be verified to ensure that no write or copy computation has taken place. If corruption is detected the operation can be repeated until satisfactory. The present invention is not concerned with such techniques and is additional to them, where provided. However, such inbuilt techniques can be used as the basis for determining whether the write operation has been performed satisfactorily in order to write the appropriate information into the second region of memory. Thus, for example, if data is successfully written to an ICC with inbuilt write verification techniques present then the conclusion of the write process can be taken as indication of a satisfactory write to allow appropriate data to be written to the second region of memory.

The type of non-volatile memory currently used in most smart cards is electrically erasable programmable read-only memory (EEPROM) and the invention is applicable particularly, but not exclusively to this. As far as reading and writing procedures are concerned EEPROM is generally divided into pages and reading or writing is carried out on one page only at a time. It can be expected that a transient writing error may corrupt the contents of one page but not others. Accordingly, it is preferred that the first and second regions are on different pages.

The invention allows the non-volatile memory to record whether there is an outstanding write error on the device and to take action accordingly when the device is used again, on application of a reset signal. Generally the protocol ISO 7816 is used, which governs the nature of reset, answer-to-reset, power and clock signals etc. If the fault is transient, the reset signal may be applied immediately so that an interrupted transaction may be resumed. If not, the reset signal is applied next time an attempt is made to use the device. Preferably, in accordance with an aspect of the invention there is provided a method of utilisation of an integrated circuit device to which data has been written as described above, the device including in the non-volatile memory an application program which controls the microprocessor to run a particular application under normal circumstances, the utilisation method including the step of initially reading the second portion of the non-volatile memory to derive write status information therefrom and, if the write status information indicates an incomplete write operation, by-passing said application program.

Thus, the action effective when an outstanding write error is present on a smart card (for example) may be to render the card useless by continued failure to run the application program. This is software invalidation of the card. Alternatively, a hardware invalidation is possible by providing an overload current to a fuse link in the card, thus blowing the fuse and rendering the card invalid. However, card invalidation is wasteful and preferably the method of utilisation includes, on detection of an incomplete write operation, a procedure of data recovery effective to restore the device to a condition in which the last data write is correct and the status information in the second region of memory reflects this. Should the data recovery procedure fail, then the above-mentioned software or hardware steps of invalidating the card may be taken.

As non-exhaustive examples of the way in which the invention may be used, three specific methods are proposed.

METHOD 1

In accordance with this method it is provided that respective and separate regions of the non-volatile memory are allocated as:

(a) a sequence register which is said second region of memory;

(b) a data copy buffer;

(c) a size register; and

(d) an address register and allocating a region of RAM or non-volatile memory as (e) a data incremental buffer, the first region of non-volatile memory being identified in size and address by data written in memory regions (c) and (d), the method of writing consisting in:

1. ensuring that the buffer (e) contains a valid data increment;
2. placing a copy of data to be updated in the buffer (b);
3. incrementing the register (a);
4. incrementing the data at the first region of memory by the amount in buffer (e) and writing the incremental amount to the first region of memory; and
5. incrementing the sequence register (a).

With this method the recovery procedure, when the register (a) indicates recovery is necessary, consists in copying the original (unamended) data from buffer (b) to said first region of memory. This restores the situation to the position before the faulty write operation.

METHOD 2

In this method it is provided that respective and separate regions of the non-volatile memory are allocated as:

(f) a write in progress flag register, which is said second region of memory;

(g) a workspace pointer register;

(h) a size register; and

(i) a data pointer register and allocating a region of RAM or non-volatile memory as (j) a new data pointer register, the first region of non-volatile memory being identified in size and position by data written in memory regions (g) and (h), the method of writing consisting in:

1. setting a workspace pointer in register
(g) to the address of non-volatile memory workspace sufficient to hold a contiguous data set corresponding to a size set in register (h);
2. copying to the workspace a copy of new data identified in address by the new data pointer at (j) and in size by the size data at (h);
3. setting the write in progress flag at (f);
4. setting an address in data pointer register (i) to the address of the work-space; and
5. clearing the write in progress flag in register (f).

Here, the recovery procedure comprises repetition of the last two steps (4 and 5), since an error would indicate that the data pointer register had not been properly written.

METHOD 3

In this method it is provided that respective and separate regions of the non-volatile memory are allocated as:

(k) a state flag register which is said second region of memory;

(l) a size register;

(m) an address register; and

(n) an update copy buffer the first region of non-volatile memory being identified in size and position by data written in registers (l) and (m), the method of writing consisting in:

1. copying new data to be written into buffer (n);
2. setting the state flat in register (k);
3. writing said new data to be written to said first region of non-volatile memory; and
4. clearing the state flag in register (k).

Here, new data is typically written directly from RAM and a copy is taken for the update copy buffer (n). If recovery is required, since it is the new data which is held in reserve in (n), the recovery procedure copies this to the required address in EEPROM (for example).

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will further be described with reference to the accompanying drawings, of which:

FIG. **1** is a schematic diagram of a smart card having EEPROM organised to effect a first method of data writing and recovery in accordance with the invention;

FIGS. **2***a*–**2B** are a flow diagram in respect of the method used in the card of FIG. **1**;

FIG. **3** is a schematic diagram similar to FIG. **1** but in respect of a second method of data writing and recovery in accordance with the invention;

FIGS. **4A**–**4B** are a flow diagram in respect of the second method;

FIG. **5** is a schematic diagram similar to FIGS. **1** and **3** but in respect of a third method of data writing and recovery in accordance with the invention; and

FIGS. **6A**–**6B** are a flow diagram in respect of the third method.

DETAILED DISCUSSION OF PREFERRED EMBODIMENTS

Referring to FIG. **1** there is shown a smart card **1** which has an interface **2** comprising a set of contacts **3** for making contact with a terminal unit **4**. In accordance with the protocol of ISO 7816 the terminal unit provides power, clock signals, a reset signal and serial data signals to the card. The card is an ICC device which includes a microprocessor **5**, RAM **6**, and EEPROM **7**.

The EEPROM **7** is divided into a set of pages **8** and is loaded with an operating system program OS, an application program AP and has a data region DR which holds data which may be read and rewritten.

A first example of the present invention is designated METHOD **1**, which is for incremental updating of data in EEPROM. In accordance with this method respective and separate regions of the data region DR of EEPROM are allocated as:

(a) a sequence register;

(b) a data copy buffer;

(c) a size register; and

(d) an address register.

A region of RAM is allocated as (e) a data incremental buffer, although this could alternatively be in EEPROM also.

Referring now to FIG. **2**(*a*) there is shown a flow diagram for the writing of data in accordance with METHOD 1. The steps include:

1. ensuring that the buffer (e) contains a valid data increment (at **9**);
2. identifying the EEPROM data to be updated (original data) by reference to the size and address registers (c), (d), giving the original location (at **10**);
3. copying the original data to buffer (b) (at **11**);
4. incrementing the sequence register (a) (at **12**);
5. calculating the new data in RAM by reference to the original data and the data in the data increment buffer

(e) and write the new data back to the original location in EEPROM (at **13**); and

6. incrementing the register (a) (at **14**).

EEPROM is such that its stored data can be corrupted if, whilst the content of the EEPROM is being changed, the power line, or the clock signal are interrupted. With the arrangement described above, data security is provided by the use of the data copy buffer in conjunction with the sequence register. By virtue of internal write verification procedures it can be assumed that if the operating system indicates completion of the write procedure **13** then the written information is in order and the sequence register (a) can be updated appropriately. If the write operation is interrupted by power line or clock signal disruption, for example, then the sequence register remains in its former state which is not appropriate to the attempted write.

In accordance with an aspect of the invention there is a check and recovery procedure available when the card receives the reset signal at any time. FIG. **2**(*b*) illustrates this. On reset at **15** the sequence register is checked at **16** to determine whether a write failure is indicated. If not then the application program AP (FIG. **1**) is executed at **17**. If failure is indicated then the original data before the last attempted write operation, which is held in data copy buffer (b) is copied to the original data address (c), (d). This step is shown at **18**. The situation before the attempted write operation is thus restored.

This method is adapted to a multi-stage operation procedure and in practice data will be fed back and forth to the terminal by a serial interface in multiple stages. The sequence register holds information as to the stage in the sequence where interruption takes place. If the original interconnection to the terminal pertains and the operation sequence can be resumed then a re-synchronisation procedure takes place and at **19** there is a check to determine whether copying/re-synchronisation has succeeded. If so then the application program AP is run. If not the software must decide from the state of the sequence register how to re-synchronise the on-card application software and the software communicating with the smart card via the serial line. If data cannot be retrieved from the data copy buffer, and the sequence register indicates that this data should be available, then the smart card is unusable, as indicated at **20**.

This may be by virtue of continued failure to implement the application program or positive steps may be taken to invalidate the card as, for example, by blowing an inbuilt fuse.

The data copy buffer (b) and the data increment buffer (e) must both be large enough to hold the largest possible data block that will be written to EEPROM using this method. An extra 5 bytes of storage are also required (size=2 bytes, address=2 bytes, sequence register=1 byte [at least]). If size can never be greater than **255**, then it can be stored in a single byte.

Since the card operates on only one page 8 (FIG. **1**) at a time in writing, security is enhanced by ensuring that separate EEPROM pages (3 in total) are used for the data copy buffer, the data increment buffer and for the rest of the additional data.

Using this method of writing to EEPROM, the number of bytes actually written to EEPROM is doubled even if a recovery is not invoked (because a copy of the original data must be stored in the data copy buffer before the EEPROM write commences). The total overhead is actually slightly more than this as size, address, and sequence register information must also be written to EEPROM.

Referring now to FIG. **3** there is shown the EEPROM configuration for a smart card (otherwise similar to that of FIG. **7**) to use a METHOD 2 in accordance with the invention. Here respective and separate regions of EEPROM (on respective pages 8) are allocated as:

(f) a write in progress flag register;

(g) a workspace pointer register;

(h) a size register; and

(i) a data pointer register.

In RAM there is allocated a region (j) as a new data pointer register. Alternatively this may also be in EEPROM.

A flow chart for the writing procedure in METHOD 2 is shown in FIG. **4**(*a*). This includes the steps of:

1. setting a workspace pointer in register (a) to the address of a workspace in EEPROM sufficient in size to hold a contiguous data set corresponding to a size set in register (h) (at **21**);
2. copying to the workspace a copy of new data in a region in RAM or EEPROM identified in size by register (h) and in position by register (i) (at **22**);
3. setting the write in progress flat (f) (at **23**);
4. setting the address in register (i) to the workspace address (at **24**); and
5. clearing the write in progress flag in register (f).

The check and recovery procedure for METHOD 2 is shown in FIG. **4**(*b*). On reset at **25** the write in progress flag is checked at **26**. If cleared the application program AP is run at **27**. If not then the last two steps (**4** and **5**) of the write procedure are repeated. Thus, the data pointer (i) is set equal to the workspace pointer (g) at **28** and the write in progress flag (f) is cleared at **29**. If this write procedure succeeds (check at **30**) the program AP is executed. If not, then the smart card is unusable (at **31**).

If an area of EEPROM is found where an EEPROM write cannot be completed, then this method readily allows the smart card application software to mark this area as unusable (permanently), and choose another area for data storage. This can greatly extend the life of the smart card (which will very probably be limited by the maximum possible number of EEPROM writes that the smart card is capable of performing), however this is at the expense of maintaining a pointer (a 2 byte overhead) to each data structure stored in EEPROM.

Under normal conditions, the Write in Progress flag is only set for the time required to update a pointer in EEPROM. This is the minimum possible theoretical update time, which should help to ensure that the recovery mechanism is invoked only very rarely. This minimises the number of attempted writes to EEPROM, and thus extends the life of the smart card.

Each data structure written to EEPROM using this method will be extended by two bytes, as a pointer to the data must be continuously maintained. There is a small overhead on each EEPROM read as all data which uses this method must be accessed via a pointer.

The EEPROM pointed to by the Workspace Pointer must be large enough to hold the largest possible data structure that will be written to EEPROM using this method. This space is only required until the EEPROM write has been successfully completed, at which point an equivalent length of EEPROM storage (which used to contain the original data) is released. An extra 7 bytes of storage are also required (Write in Progress flag=1 byte, New Data Pointer =2 bytes, Workspace Pointer=2 bytes, Size =2 bytes). If Size can never be greater than 255, then it can be stored in a single byte.

Using this method of writing to EEPROM, the data structure is only written to EEPROM once, but three point-

ers have to be updated (the New Data Pointer, the Workspace Pointer and the Data Pointer—in that order). The Size, Address and Sequence Register information must also be written to EEPROM.

Referring now to FIG. 5 there is shown EEPROM allocation for a METHOD 3 of implementing the invention. It is to be understood that the EEPROM of FIG. 5 is incorporated in a smart card otherwise similar to that of FIG. 1. In FIG. 5, separate regions of EEPROM (on separate pages 8) are allocated as:

(k) a state flag register;

(l) a size register;

(m) an address register; and

(n) an update copy buffer.

The writing procedure in METHOD 3 is illustrated in FIG. 6(*a*). The following steps are implemented:

1. Copy new data into buffer (n) (at 32);
2. Set state flag (k) (at 33);
3. Copy the new data to EEPROM region identified by size (l) and address (m) (at 34); and
4. Clear state flag (k) (at 35).

The check and recovery procedure illustrated in FIG. 6(*b*) has reset at 36, and a check for the setting of state flag (k) at 37. If the flag is not set then application program AP is run at 38. Otherwise the new data residing in buffer (n) is copied to the region (l), (m) at 39 and the state flag (k) is cleared at 40. If successful, the application program is run. If not, the card is useless (41).

An additional Data area (buffer n) must be large enough to store the largest amount of data which will be written to EEPROM, plus 5 bytes (Size=2 bytes, Address=2 bytes, State Flag—1 byte). If Size can never be greater than 255, then it can be stored in a single byte.

Using this method of writing to EEPROM, the number of bytes actually written to EEPROM is doubled even if a Recovery is not invoked (because a copy of the data must be written to EEPROM). The total overhead is actually slightly more than this as Size, Address must also be written to EEPROM.

To be able to tell that data has not been altered, error detection techniques must be implemented. Error detection usually comprises calculating a checksum whenever the data is updated, storing this checksum, and verifying that it is correct during every subsequent data read. The actual method used to calculate the error detection checksum is irrelevant for the purposes of this document, indeed some smart cards have error detection processes built into the EEPROM hardware, and their particular method of operation may well not be known.

An EEPROM write is deemed to be complete only when the error detection system has been appropriately updated, and has been verified correctly.

Each byte of EEPROM can only be changed a finite number of times before it ceases to function correctly. This is typically 10^5 to 10^6 write cycles. Therefore there is a finite chance of data being altered whilst it resides in EEPROM, and an EEPROM read must only be accepted as valid if the error detection system verifies that the data has not been altered. If an error is detected during an EEPROM read, it probably means that one or more bytes in the smart card's EEPROM have reached the end of their active life.

Using one of the methods of writing to EEPROM described above ensures that error correction (as opposed to error detection) is not required. Either the EEPROM operation takes place successfully, or the smart card is unusable. There are no circumstances in which an error needs to be corrected. This simplifies the software and reduces the data storage requirements, as error correction is computationally intensive and requires more dedicated bytes of storage than error detection.

One of the three methods of writing data to EEPROM described above (Method Number 1) explicitly keeps a counter (Sequence Register) which stores knowledge of the last successful operation in the series of operations performed during writing to EEPROM. Methods 2 and 3 may have, but do not explicitly require a counter of this type as they reply upon flags which hold information showing whether or not writing to EEPROM has successfully completed.

Even though a method of writing to EEPROM does not always explicitly require a numeric counter, it should be clearly noted that in many systems it will be necessary to maintain such a counter so that interrupted processes of any kind can be restarted. It is of course vitally important for this counter to be written to EEPROM in a secure manner, as if it is not correct it cannot be relied upon by smart card application software attempting to restart an interrupted process.

We claim:

1. A method of utilization of an integrated circuit device, the device having an interface for temporary connection to a terminal unit; a microprocessor; random access memory (RAM) and non-volatile memory, the method of utilization including:

a method of writing data to said non-volatile memory comprising:

allocating a first region of the non-volatile memory for data to be written,

allocating a second region of non-volatile memory for write status information,

performing a data write operation to write data to said first region, and

writing information to said second region signifying a valid data write if, and only if, the data write operation is performed completely, and

the method of utilization further including the method of responding to a reset of the device by the steps of:

initially reading the said second region of the non-volatile memory to derive write status information therefrom and,

if the write status information indicates an incomplete write operation, enabling invalidation of the integrated circuit device.

2. A method of utilization of an integrated circuit device as claimed in claim 1 including the step of instituting recovery of data to the non-volatile memory, invalidation of the integrated circuit device being effected only on failure of said recovery.

3. A method of utilization of an integrated circuit device as claimed in claim 1 wherein the non-volatile memory includes an application program which controls the microprocessor to run a particular application under normal circumstances and invalidation of the integrated circuit device is software invalidation whereby said application program is by-passed.

4. A method of utilization of an integrated circuit device as claimed in claim 1 wherein invalidation of the integrated circuit device is effected by incapacitating the hardware of the device.

5. A method of utilization of an integrated circuit device as claimed in claim 1 wherein the non-volatile memory is divided into pages and write operations are performed on only one page at a time, the first and second regions of memory being on different pages.

6. A method of utilization of an integrated circuit device as claimed in claim 1 wherein the non-volatile memory is electrically erasable programmable read-only memory (EEPROM).

7. A method of utilization of an integrated circuit device as claimed in claim 1 wherein said second region of memory is a status register, said status information is indicative of the last satisfactorily performed stage of a multi-stage operation sequence and said data recovery procedure is effective to recover the multi-stage operation sequence from the stage at which it failed, as indicated by the status register.

8. A method of utilization of an integrated circuit device as claimed in claim 7 wherein respective and separate regions of the non-volatile memory are allocated as:

a. a sequence register which is said second region of memory;

b. a data copy buffer;

c. a size register; and

d. an address register and allocating a region of RAM or non-volatile memory as (e) a data incremental buffer, said first region of non-volatile memory being identified in size and address by data written in memory regions (c) and (d), said method of writing comprising:

1. ensuring that the buffer (e) contains a valid data increment;

2. placing a copy of data to be updated in the buffer (b);

3. incrementing the register (a);

4. incrementing the data at the first region of memory by the amount in buffer ((e) and writing the incremental amount to the first region of memory; and

5. incrementing the sequence register (a).

9. A method of utilization of an integrated circuit device as claimed in claim 8 wherein the recovery procedure includes copying the data from the data buffer (b) to said first region of memory.

10. A method of utilization of an integrated circuit device as claimed in claim 1 wherein said second region of memory is a flag region and said status information is a flag which is set if said write operation is verified as satisfactory and which is otherwise not set.

11. A method of utilization of an integrated circuit device as claimed in claim 10 wherein respective and separate regions of the non-volatile memory are allocated as:

f. a write in progress flag register, which is said second region of memory;

g. a workspace pointer register;

h. a size register; and

i. a data pointer register and allocating a region of RAM or non-volatile memory as (i) a new data pointer register, said first region of non-volatile memory being identified in size and position by data written in memory regions (g) and (h), said method of writing comprising:

1. setting a workspace pointer in register (g) to the address of non-volatile memory workspace sufficient to hold a contiguous data set corresponding to a size set in register (h);

2. copying to the workspace a copy of new data identified in address by the new data pointer at (j) and in size by the size data at (h);

3. setting the write in progress flag at (f);

4. setting an address in data pointer register (i) to the address of the workspace; and

5. clearing the write in progress flag in register (f).

12. A method of utilization of an integrated circuit device as claimed in claim 11 wherein the recovery procedure comprises the steps of setting the address in data pointer register (i) to the address of the workspace and clearing the write in progress flag in register (f).

13. A method of utilization of an integrated circuit device as claimed in claim 10 wherein respective and separate regions of the non-volatile memory are allocated as:

k. a state flag register which is said second region of memory;

l. a size register;

m. an address register; and

n. an update copy buffer said first region of non-volatile memory being identified in size and position by data written in registers (l) and (m), said method of writing comprising:

1. copying new data to be written into buffer (n);

2. setting the state flag in register (k);

3. writing said new data to be written to said first region of non-volatile memory; and

4. clearing the state flag in register (k).

14. A method of utilization of an integrated circuit device as claimed in claim 13 wherein the recovery procedure comprises the steps of copying the contents of the update copy buffer (n) to said first region of non-volatile memory identified by the contents of the registers (l) and (m) and clearing the flag in register (k).

15. An integrated circuit card (ICC) device comprising:

an interface for temporary connection to a terminal unit;

a microprocessor responsive to said interface;

a random access memory (RAM) connected to said microprocessor; and

a non-volatile memory connected to said microprocessor, said non-volatile memory further including:

a first region for storing data to be written,

a second region for storing write status information, said microprocessor including programming:

for performing a data write operation to write data to said first region;

for performing a data write operation to said second region signifying a valid data write status if, and only if, the data write operation to said first region is performed completely;

for reading, in response to a reset of the device, said second region of the non-volatile memory to derive write status information therefrom; and

enabling, if the write status information indicates an incomplete write operation, invalidation of the integrated circuit device.

* * * * *

US006253370B1

(12) **United States Patent**
Abadi et al.

(10) **Patent No.: US 6,253,370 B1**
(45) **Date of Patent: *Jun. 26, 2001**

(54) **METHOD AND APPARATUS FOR ANNOTATING A COMPUTER PROGRAM TO FACILITATE SUBSEQUENT PROCESSING OF THE PROGRAM**

(75) Inventors: **Martin Abadi**, Palo Alto; **Sanjay Ghemawat**, Mountain View; **Raymond Paul Stata**, Palo Alto, all of CA (US)

(73) Assignee: **Compaq Computer Corporation**, Houston, TX (US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/982,088**

(22) Filed: **Dec. 1, 1997**

(51) **Int. Cl.**[7] **G06F 9/445**

(52) **U.S. Cl.** **717/5**; 717/2; 717/9; 713/182; 713/200

(58) **Field of Search** ..... 395/705, 709; 717/5, 9, 2; 713/20 D, 182

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,418,958 * 5/1995 Goebel 395/709
5,734,822 * 3/1998 Houha et al. 395/200.6

OTHER PUBLICATIONS

Necular et al., "The Design and Implementation of a Certifying Compiler", ACM, pp. 333–344, Jun. 1998.*
Myers, "JFlow: Practical Mostly-Static Information Flow Control", ACM, pp. 228–241, Jan. 1999.*
Thorn, "Programming Language for Mobile Code", ACM Computing Surveys, vol. 29, No. 3, pp. 213–239, Sep. 1997.*
Aho et a., "Compilers, Principles, Techniques, and Tools", Addison–Wesley, pp. 10–15, 396–400, 517–518, 528–533, Mar. 1988.*
George C. Necula and Peter Lee, "Proof–Carrying Code," CMU–CS–96–165.
Peter Lee's Web Page: http://www.cscmu.edu/~petel/papers/pcc/, "Proof–Carry Code," printed Nov. 26, 1997.
Peter Lee and George C. Necula, "Research on Proof–Carrying Code for Mobile–Code Security," DARPA Workshop on Foundations for Secure Mobile Code, Mar. 26–27, 1997.
The ANDF Home Page: http://www.osf.org/andf/, "A Brief Introduction to ANDF", printed Oct. 1, 1997.

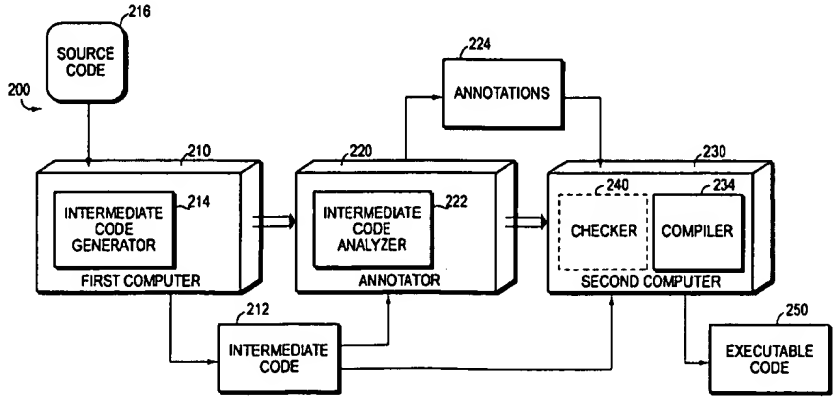
* cited by examiner

Primary Examiner—Mark R. Powell
Assistant Examiner—Ted. T. Vo
(74) *Attorney, Agent, or Firm*—Cesari and McKenna, LLP; Edwin H. Paul

(57) **ABSTRACT**

A method and apparatus annotates a computer program to facilitate subsequent processing of the program. Code representing the program is generated at a first computer system. Annotations are generated for the code that provide information about the code. At a second computer, the code is processed according to the information provided by the annotations. The annotations, for example, can indicate a control flow graph representing a flow of execution of the code. Also, the information provided by the annotations can be a register allocation that maps data structures of the code to registers of the second computer system. The second computer system can use such information to guide the interpreting of the code or to transform the code into a more optimized form. Other exemplary annotations can indicate that running the executable form of the code would perform an unauthorized operation at the second computer system. The second computer system could then reject the code instead of performing subsequent processing on the code. When the source of the annotations is untrusted by the second computer system, the second computer system can use a checker to verify the integrity of the annotations.

16 Claims, 3 Drawing Sheets



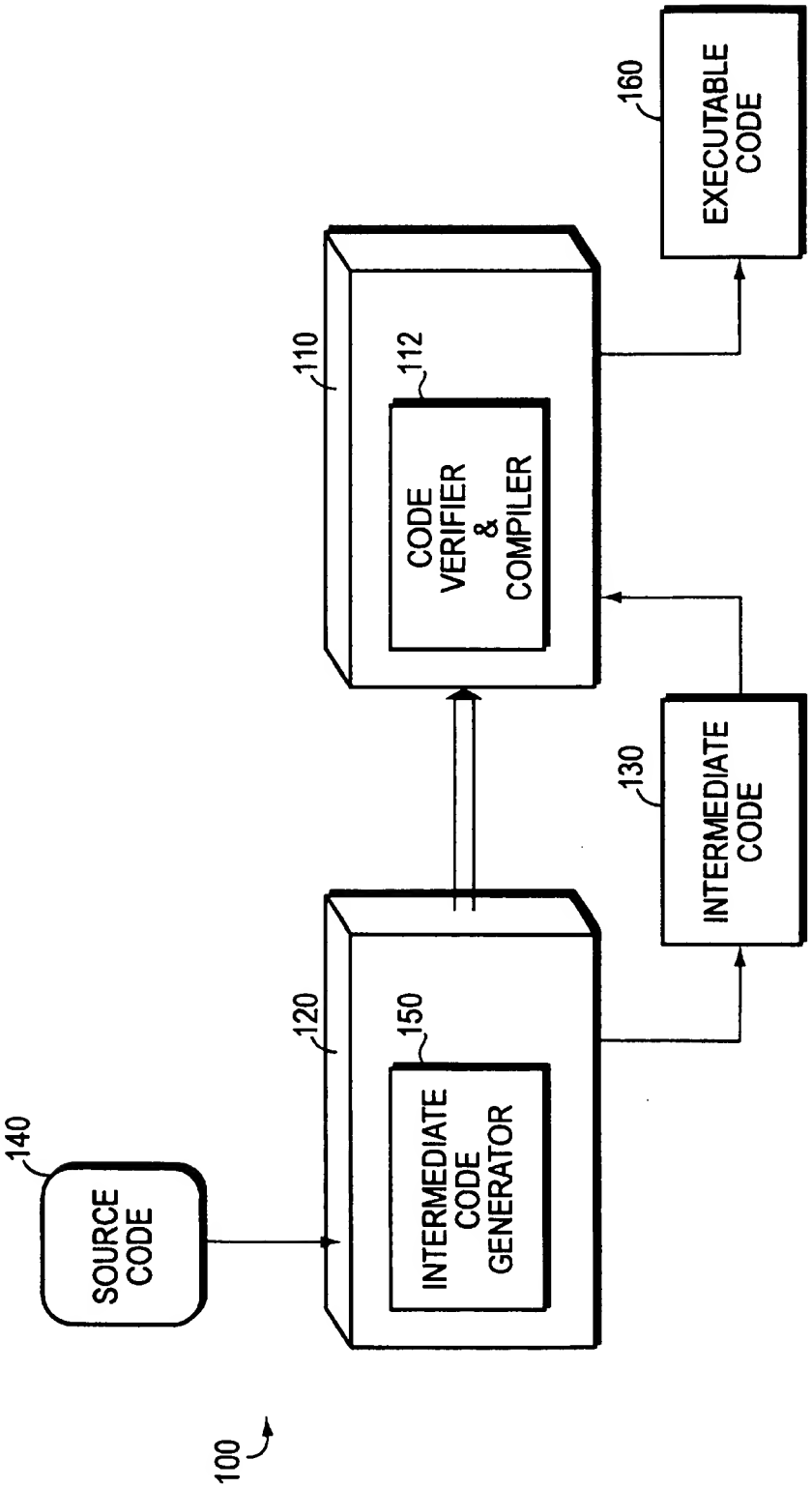


FIG. 1
(PRIOR ART)

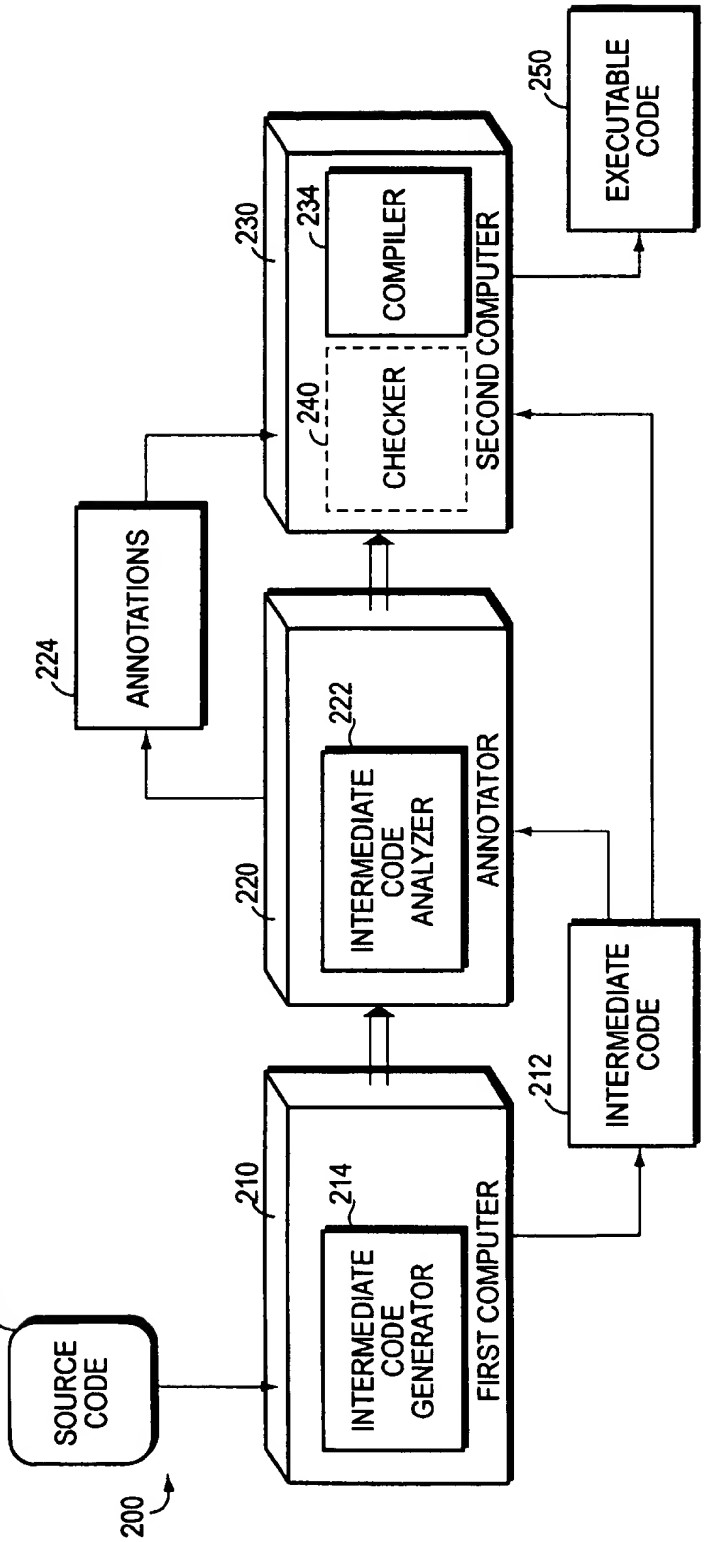

200
216
SOURCE CODE
210
214
INTERMEDIATE CODE GENERATOR
FIRST COMPUTER
212
INTERMEDIATE CODE
220
222
INTERMEDIATE CODE ANALYZER
ANNOTATOR
224
ANNOTATIONS
230
234
COMPILER
240
CHECKER
SECOND COMPUTER
250
EXECUTABLE CODE

FIG. 2

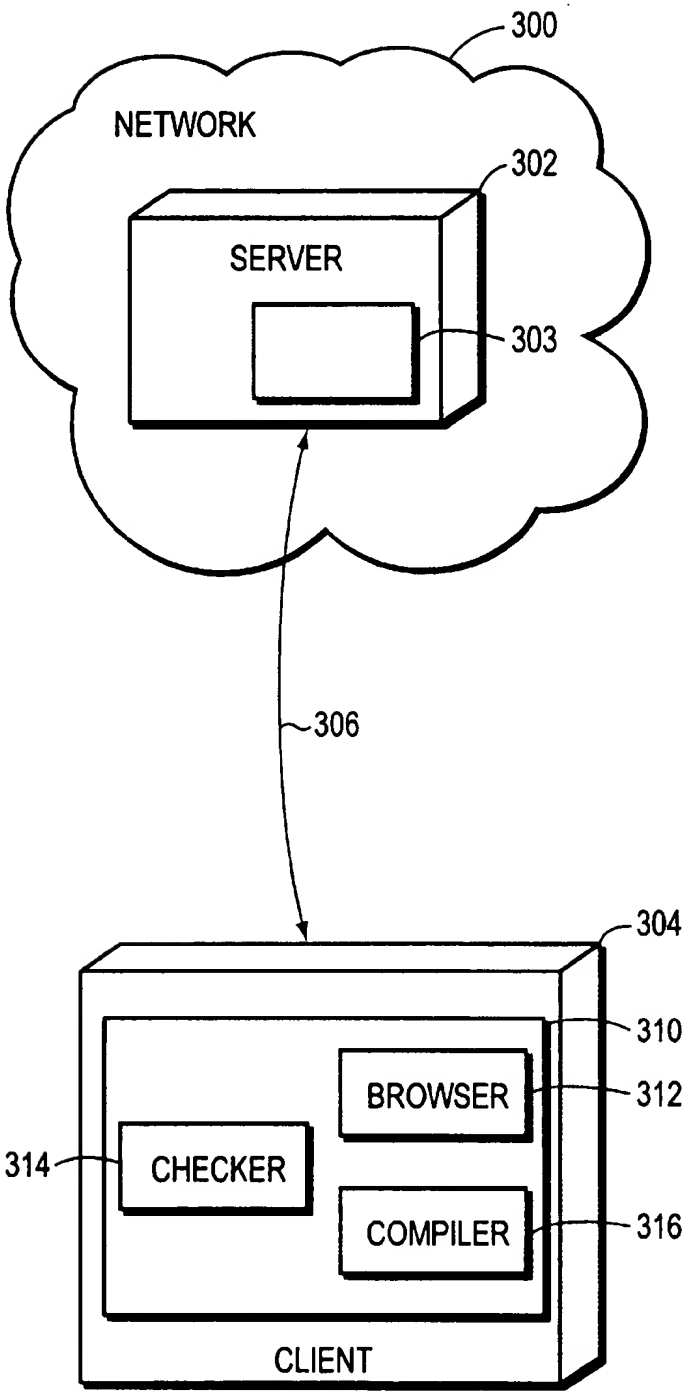

300
NETWORK
302
SERVER
303
306
304
310
BROWSER
312
314
CHECKER
COMPILER
316
CLIENT

FIG. 3

METHOD AND APPARATUS FOR ANNOTATING A COMPUTER PROGRAM TO FACILITATE SUBSEQUENT PROCESSING OF THE PROGRAM

FIELD OF THE INVENTION

This invention relates generally to the processing of mobile, computer programs, and more particularly to annotating such programs to assist downstream processing phases.

BACKGROUND

In computer systems, and particularly in networked computer systems, computers commonly acquire programs to execute from other computers. Before executing an acquired program, the acquiring computer typically performs processing on the program. For example, the computer may compile the program into machine language native to that computer. As another example, the computer may verify that the program satisfies certain security constraints. This verification is particularly important because, generally, the computer distrusts the acquired program; the security checks ensure that the program does not tamper with files and other resources of the computer.

FIG. **1** illustrates a typical prior art network **100** in which a first computer **110** uses a program processing tool **112** to verify and compile a program downloaded from a second computer **120**. The program downloaded from the second computer **120** is in an intermediate form **130** that represents the program. The second computer **120** used an intermediate code generator **150** to generate the intermediate form **130** from source code **140** of the program. At the first computer **110**, the processing tool **112** analyzes the code **130** to determine whether the code **130** is safe to compile and execute. The tool **112** also performs code optimization techniques to produce executable machine code **160** native to the first computer **110**.

Security checks and compiler analyses consume system time and, as a result, can reduce performance. These analyses can also be ineffective because of insufficient information to perform a proper security check or insufficient time to thoroughly process available information.

Security checks, for example, may err on the side of caution and reject secure code because the information necessary to prove that the code is secure is lacking. Moreover, a security check itself may be a source of vulnerability because it is incorrectly designed or improperly implemented. Unwittingly, this security check may leave open doors for attack. Also, some compilers, such as just-in-time compilers, may not have sufficient time to perform thorough analysis for optimization. Without enough time for optimization, the machine code may perform poorly.

As a result, a need remains for a method and an apparatus that facilitate security checks and code analyses. Such a method and apparatus can lead to improved accuracy of the security checks and to machine code that performs better than what can currently be generated.

SUMMARY

In accordance with the present invention, an objective is to enhance program code, such as mobile code, with supplementary information that will help subsequent processing stages. Having such information available during subsequent processing stages will, for example, lead to more accurate determinations of the security of the code and to improved performances of generated machine code.

A method performed according to the principles of the invention achieves the aforementioned and other objectives when processing intermediate code generated at a first computer system by generating annotations for the code. The annotations provide information about the intermediate code that can be used to process the code. A second computer system receiving the code and the annotations can then process the code according to the information provided by the annotations.

The annotations, in general, provide information that is useful to the second computer system for processing the code. For example, the annotations can be a control flow graph that represents an execution flow of the code. Also, the annotations can provide a register allocation that maps the data structures of the code to machine registers of the second computer system. Other annotations can provide method offsets. Such information provided by the annotations can be useful to the second computer system, for example, when interpreting or compiling the code. As yet another example, the annotations can indicate whether running the code would perform unauthorized operations on the second computer system.

These annotations can be generated at a number of locations in a network before being transmitted to the second computer system. For example, the first computer system that produced the code can also produce the annotations and send both the code and the annotations to the receiving computer system. The first computer system may produce the code and the annotations concurrently or produce the annotations after the code has been generated. Also, the first computer system may add the annotations to the code and send both together to the second computer system, or store the annotations separately from the code and transmit the annotations and code separately. In still another example, a third computer system between the first and second computer systems, for example, a computer on a firewall protecting the second computer system from receiving potentially harmful programs, can generate and transmit the annotations to the second computer system.

Just as code from the first computer cannot always be trusted, downloaded annotations should also not be trusted unless a trusted system, such as the aforementioned third computer system on the firewall, generated the annotations. When the annotations come from an untrusted system, the second computer system must check the correctness of the annotations that the second computer system uses. Checking the analysis provided by the annotations, however, is often faster and simpler than performing the analysis, so the invention still improves the performance and reduces the vulnerability of the second computer system.

In terms of the disclosed apparatus, the invention comprises a first computer system and a second computer system coupled to each other by a network. The second computer system requests a computer program from the first computer system. An annotator generates an annotation for the program. The annotation provides information about the program that characterizes the program. The second computer system receives the code and the annotation and processes the code according to the information provided by the annotation.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram schematic of an embodiment of the present invention;

FIG. 2 is another more detailed block diagram schematic of an embodiment of the present invention; and

FIG. 3 is a block diagram of an exemplary application of the present invention

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. 2 shows an exemplary networked computer system **200** including a program annotator **220** coupled to a first computer **210** and a second computer **230**. For purposes of illustration, the user of the second computer **230** does not trust the first computer **210** or any code coming from the first computer **210**. This means that the user of the second computer **230** does not know whether an executable form of the code **212** will perform any unauthorized operations, such as accessing files and directories of the second computer **230**. Accordingly, the user of the code **212** should verify the integrity of untrusted code before executing it. The first computer **210** includes an intermediate code generator **214** that converts source code **216** of a computer program into an intermediate code **212**. The source code **216** can be written in any programming language, such as Java, C or C++, but the intermediate code generator **214** must be able to process the semantics and syntax of that programming language in order to produce the intermediate code **212**. The intermediate code **212** produced by the generator **214** is machine-independent, that is, the code **212** itself does not run on any particular computer without further processing, e.g., interpreting or compiling. It is to be understood that the practice of the principles of the invention is not limited to intermediate code, but rather that annotations can be generated for various other types of code, such as, for example, source code, machine code, machine-dependent or machine-independent code, high-level or low-level code, assembly code, etc.

The annotator **220** includes an intermediate code analyzer **222** that analyzes the intermediate code **212** from the first computer **210** and produces annotations **224** as a result. This analysis can include, for example, mapping variables to registers, determining a control flow of the code **212**, determining methods for optimizing the code **212**, checking that all data structures are initialized and that the code **212** is syntactically well-formed, contains valid references to data structures, data fields, and other code, and verifying that operations performed by the code **212** do not underflow or overflow the stack. These examples are simply illustrative.

From the annotator **220**, the intermediate code **212** and the annotations **224** pass to the second computer **230**. Although shown in FIG. **1** to be separately forwarded to the second computer **230**, the intermediate code analyzer **222** can annotate the code **212** so that the annotations **224** are placed in the code **212**, producing an annotated intermediate code. As a result, the code **212** and the annotations **224** arrive concurrently at the second computer **230**.

Placing the annotations **224** in the code **212** displaces the need for locally caching the analysis. Before the present invention, each user of the intermediate code **212** would store the analysis performed on the code **212** for subsequent use. This way, the computer would not have to repeat the analysis each time the intermediate code **212** was downloaded. With local caching, however, only the computer with the cached analysis benefited from that analysis. Using the present invention, the analysis that is recorded by the annotations **224** in the intermediate code **212** can benefit any user with access to the annotated intermediate code.

The annotator **220** can reside at the first computer **210** or at a third computer (not shown) connected to both the first and second computers **210, 230**. Conceivably, the annotator **220** could reside at the second computer **230**, but the benefits of annotating are greater when the intermediate code **212** arrives at the second computer **230** already annotated.

Normally, it would be easier to annotate the intermediate code **212** at the same computer where the intermediate code **212** is produced because of the availability of the original source code **216**. For example, when the annotator **224** resides at the first computer system **210**, the code and the annotations **224** can be produced concurrently, or the annotations **224** can be produced after the code has been generated. Having the annotator **220** reside at the first computer **210**, therefore, produces advantages. On the other hand, the annotations **224** produced by the first computer **210** are untrusted because the first computer **210** is untrusted. Thus, the second computer **230** should verify the integrity of the annotations **224**.

For this purpose, the second computer **230** has a checker **240** for verifying the integrity of the annotations **224**. Because it is often faster and simpler to check annotations than to produce annotations, the advantages of annotating at an untrusted system remain. The checker **240** can immediately reject the code **212** when the checker **240** determines that the annotations **224** are invalid. Invalid annotations **224** include those annotations that present a false representation of the operation of the code **212** or perform operations that are unauthorized by the second computer **230** or are not well-formed, i.e., fail to follow a particular format. Conversely, valid annotations **224** are well-formed and accurately reflect the operation of the code **212**. The checker **240**, then, can quickly conclude from the annotations **224** whether the intermediate code **212** should be subsequently processed, e.g., interpreted or compiled.

The dashed lines in FIG. **2** indicate that the second computer **230** may not need a checker **240** when the annotations **224** come from a trusted source. An example of a trusted source is a third computer (not shown) at a firewall between the first computer **210** and the second computer **230**, protecting the second computer **230** from harmful programs. The annotator **220** can reside at this third computer and produce annotations **224** that are trusted by the second computer **230**.

The second computer **230** includes a compiler **234** for transforming the intermediate code **212** into executable machine code **250**. The machine code **250** is dependent on the microprocessor running the second computer **230**. The compiler **234** has added capabilities for handling the format of the annotations **224** and for using the annotations **224** as guidance during construction of the machine code **250**. For example, the additional capabilities of the compiler **234** include analyzing the annotations **224** and rejecting the intermediate code **212** when the annotations **224** indicate that the code **212** is not secure. The compiler **234** can also reference the annotations **224** to optimize the machine code **250**. Alternatively, the second computer **230** can include an interpreter capable of using the annotations to determine whether to execute the intermediate code **212** and then for guidance during any subsequent code execution.

ANNOTATIONS

In general, the annotations **224** produced by the analyzer **222** include any information about the code **212** that can be obtained from static analysis. This information facilitates subsequent processing of the code **212**. The annotations **224** that provide information about the code **212** are various and fall into at least two types: annotations that characterize

properties of the code; and annotations that are in the form of a formal proof of the code. This categorizing of annotations is not intended to be exhaustive, but rather to distinguish annotations that characterize properties of the code from annotations that are a proof of the code.

The first type of annotations **224**, those that characterize properties of the code **212**, provide the second computer **230** with information that assists in a wide variety of subsequent processing of that code **212**. Such subsequent processing includes determining whether the code is safe for additional subsequent processing, such as executing machine code, or interpreting or compiling intermediate code. For example, when the code **212** is in machine code form, this type of annotations **224** contains information about how the code accesses memory, allowing the second computer **230** to conclude that this machine code is safe to execute, or such annotations **224** can contain information about what registers are live at different program points, allowing the code to be optimized for increased performance. Alternatively, when the code **212** is in an intermediate code form, the annotations can provide useful information for optimally interpreting the intermediate code or transforming the intermediate code into an executable form.

The information provided by these annotations can range from a detailed description of a particular property of the code to a broad, overall perspective of the entire code **212**. For instance, exemplary annotations can characterize the behavior of a single code statement, a block of code statements, or the flow of execution of the entire code **212**. The following examples are illustrative of the diversity and uses of annotations that characterize properties of the code. Any one or all of these exemplary annotations may be generated for the code **212** and used by the second computer **230** as aid in the subsequent processing of the code **212**.

Exemplary annotations **224** of the first type can indicate what variables are used in the code **212** and the types of values stored in those variables. The particular annotation for the code statement

"X_1:=0",

for example, can be

{X_1: integer, X_2: undefined},

where X_1 and X_2 are the two variables used by the intermediate code **212**. This particular exemplary annotation indicates that at this point in the code **212**, the variable X_1 holds a data structure of an integer type, while the type of the data structure in X_2 is undefined. Such annotations **224**, for example, can simplify and accelerate for the second computer **230** the task of type-checking data structures of the code **212** to determine whether the intermediate code **212** is secure for subsequent execution. Thus, the second computer **230** can determine beforehand that run-time checks of the intermediate code **212** are unnecessary. As another exemplary use of such annotations, the information about the data types can assist run-time optimization by enabling tag-less garbage collection.

Another exemplary annotation **224** is a control flow graph that represents the flow of execution of the entire code **212**. Some exemplary annotations **224** can be less encompassing and represent the behavior of blocks of code statements. Such annotations **224** for blocks of statements can be placed at a block entry point, at an exit point, or at both points.

Other annotations **224** can map data structures to machine registers of the second computer **230**. The mapping of data structures to machine registers can help optimize machine code **250** through efficient use of the machine registers. This register allocation can benefit just-in-time compilers that commonly make sub-optimal use of the registers because of the limited time in which to analyze intermediate code **212**.

Still other annotations **224** that characterize the code **212** can provide method offsets. Method offsets direct the compiler **234** to locations within an object where the compiler **234** can find particular methods. These annotations can help the compiler **234** avoid clashes in method offsets in situations of multiple inheritance. Still others **224** may show when a level of indirection can be removed from a data structure.

Annotations of the second type provide a formal proof of some property of the code. The formal proof uses formal logic reasoning about the code. The proof assures that the code will behave according to a prescribed policy when that proof is validated. An example of the second type of annotations is described by George Necula in "Proof-Carrying Code", 1997, incorporated by reference herein. There, a compiler adds a formal proof to native binary code while the compiler produces the binary code. When the proof is validated, the binary code is deemed safe to execute.

Annotations of the second type can be used to practice the principles of the present invention. A proof provided by such annotations can be used to determine whether code should be subsequently executed, i.e., compiled or interpreted. When the proof is validated, annotations of the previously-mentioned first type can then be used to guide such subsequent execution. In general, to produce annotations, the analyzer **222** statically analyzes the intermediate code **212** like a conventional compiler. Off-loading the analyses to the analyzer **222** allows the second computer **230** to more quickly and more effectively process the intermediate code (e.g., produce better machine code **250**) than if the second computer **230** had to perform its own analyses. This is because the annotator **220** may have more time than the second computer **230** to produce a more thorough analysis. Also, the annotator **220** may have access to available source code **216**, whereas such information may not be available to the second computer **230**.

FIG. **3** illustrates an exemplary application using the principles of the present invention to process a computer program. A communication network **300** connects a server **302** in the network **300** with a client computer **304** by network link **306**. An example of such a network **300** is the Internet. The server **302** supports a web page; that is, the server **302** maintains documents, pages and other forms of data for retrieval. Applets, which are small programs compiled to an intermediate form, might be attached to the web page when the web page is retrieved.

The server **302** includes an annotator **303** that statically analyzes and annotates, according to the principles of the invention, each applet attached to the web page. That the annotator **303** statically analyzes the applet before the applet is sent to the client **304** distinguishes the present invention from known techniques, such as a Java™ virtual machine, that analyze the applet at the client **304**.

The client **304** includes memory **310** for storing a browser **312**, an annotation checker **314**, and a compiler **316**. The memory **310** can include a hard disk and main memory. The browser **312** provides a user of the client **306** with an interface that simplifies access to the server **302**. Examples of a browser are Netscape Communicator™ and Microsoft Internet Explorers™.

During an execution of the browser **312**, the client **304** can request access to the web page on the server **302**. The browser **312** issues the request to the server by the link **306**.

In response to the request, the server **302** returns the data associated with the requested web page to the client **304**. When the retrieved web page is accompanied by an attached applet, the server **302** sends the annotated intermediate code representing the applet to the client **304**.

When the server **302** is trusted by the client **304**, the client **304** can process the annotated intermediate code according to the annotations embedded in the code without having to verify the annotations. This processing can include checking the safety of the applet and executing the applet. As used in this context, "executing" means interpreting or compiling. For example, the annotations may provide typing of the variables in the code from which the browser can determine whether the applet is safe to execute on the client **304**. As another example, the annotations can suggest register allocations, for example, that help the browser execute the applet through efficient use of machine registers of the client **304**.

Typically, however, the client **304** does not trust the applet produced by the server **302**. In this event, the client **304** would analyze the annotations along with the applet to make sure that the applet would not perform any unwanted operations when the applet runs on the client **304**. The checker **314** accordingly verifies the integrity of the annotations in the applet code. The browser **312** rejects the applet when the checker **314** determines that the annotations are false. On the other hand, when the checker **314** determines that the annotations are valid, the browser **312** can process the applet, as previously noted, according to the annotations in the applet code.

Although described within the context of the Internet and web browsers, the invention can be practiced within any other context where programs are annotated to facilitate subsequent program processing stages. The foregoing description has been directed to specific embodiments of this invention. It will be apparent, however, that variations and modifications may be made to the described embodiments, with the attainment of all or some of the advantages. It is the object of the appended claims, therefore, to cover all such variations and modifications as come within the spirit and scope of the invention.

What is claimed is:

1. A computerized method for processing code representing a computer program, the code being generated at a first computer system, the method comprising the steps of:

generating an annotation for the code that characterizes at least one property of the code;

analyzing the annotation, at a second computer system, to determine whether the code can safely operate at the second computer system and to provide information for optimizing the code's operating performance; and

transforming and optimizing the code into an executable code in the second computer according to the information contained in the annotation if the analysis indicates that the code can be safely operated.

2. The method of claim **1** further comprises interpreting the code according to the information provided by the annotation.

3. The method of claim **1** wherein the annotation includes information on register allocation that maps data structures of the code to registers of the second computer system.

4. The method of claim **1** wherein the annotation includes information on a control flow graph representing a flow of execution of the code.

5. The method of claim **1** wherein the annotation includes information on a method offset.

6. The method of claim **1** wherein the annotation indicates data types of variables in the code.

7. The method of claim **1**, further comprising the step of:

verifying at the second computer system that the annotation is valid.

8. The method of claim **1** wherein the generating of the annotation occurs at the first computer system.

9. The method of claim **1** wherein the generating of the annotation occurs at a third computer system.

10. The method of claim **1**, further comprising the step of:

adding the annotation to the code to produce annotated code; and

sending the annotated code to the second computer system.

11. The method of claim **1** wherein the code is intermediate code requiring processing before the code can operate at the second computer system.

12. The method of claim **1**, further comprising the steps of determining from the information provided by the annotations whether the code can be trusted to operate at the second computer system and operating the code only if the code can be trusted.

13. The method of claim **1** wherein the code is trusted to operate at the second computer system when the annotations are generated at a trusted computer system.

14. The method of claim **1** wherein determining whether the code should be processed includes determining whether running an executable form of the code would perform an unauthorized operation at the second computer system.

15. An apparatus for processing a computer program, comprising:

a first computer system and a second computer system coupled to each other by a network, the second computer system requesting a computer program from the first computer system;

an annotator, coupled to receive the program, generating an annotation for the program, the annotation characterizing at least e property of the program; and

the second computer receiving the code and the annotation, the second computer analyzing the annotation to determine whether the code can safely operate at the second computer system and provide information for optimizing the code's operating performance, and if the analysis indicates that the code can be safely operated, the second computer system transforming and optimizing the code into an executable code in the second computer according to the information contained in the annotation.

16. A system for processing a computer program, the system comprising:

a first computer system and a second computer system coupled to each other by a network, the first computer system comprising a means for generating code;

means for generating an annotation for the code, the annotation providing information that characterizes at least one property of the code;

means for analyzing the annotation, at the second computer system, to determine whether the code can safely operate at the second computer system and to provide information for optimizing the code's operating performance; and

means for transforming and optimizing the code into an executable code in the second computer according to the information contained in the annotation if the analysis indicates that the code can be safely operated.

* * * * *

US005161231A

United States Patent [19]

Iijima

[11] Patent Number: 5,161,231

[45] Date of Patent: Nov. 3, 1992

[54] **PROCESSING SYSTEM WHICH TRANSMITS A PREDETERMINED ERROR CODE UPON DETECTION OF AN INCORRECT TRANSMISSION CODE**

[75] Inventor: **Yasuo Iijima**, Yokohama, Japan

[73] Assignee: **Kabushiki Kaisha Toshiba**, Kawasaki, Japan

[21] Appl. No.: **667,376**

[22] Filed: **Mar. 12, 1991**

Related U.S. Application Data

[63] Continuation of Ser. No. 361,349, Jun. 5, 1989, abandoned, which is a continuation of Ser. No. 97,660, Sep. 16, 1987, abandoned.

[30] **Foreign Application Priority Data**

Sep. 27, 1986 [JP] Japan 61-229149
Nov. 14, 1986 [JP] Japan 61-271207

[51] **Int. Cl.**[5] **H04L 13/00**; G06F 11/00; G06F 11/14

[52] **U.S. Cl.** **395/800**; 364/DIG. 1; 364/DIG. 2; 364/265; 364/265.1; 364/932.1; 235/380

[58] **Field of Search** ..... 395/DIG. 1, DIG. 2, 395/800; 235/380, 80, 79, 487, 449, 492; 371/33, 34, 35

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,439,859	3/1984	Donnan	371/32
4,575,621	3/1986	Dreifus	235/380
4,707,592	11/1987	Ware	235/379
4,712,214	12/1987	Meltzer et al.	371/32
4,760,514	7/1988	Hasegawa et al.	364/200
4,779,274	10/1988	Takahashi et al.	371/32

FOREIGN PATENT DOCUMENTS

0182244	5/1986	European Pat. Off. .	
0190733	8/1986	European Pat. Off. .	
0194839	8/1988	European Pat. Off. .	
0049650	4/1982	France .	
60-220645	5/1985	Japan	371/32

OTHER PUBLICATIONS

Farber et al., "Bussysteme, Parallele und serielle Bussysteme in Theorie und Proxis," Mit 150 Bildern und 29 Tabellen, 1984, pp. 104–128.

English Patent Abstract of Japan "Data Transmission System." vol. 10, No. 75 (Nov. 5, 1985).

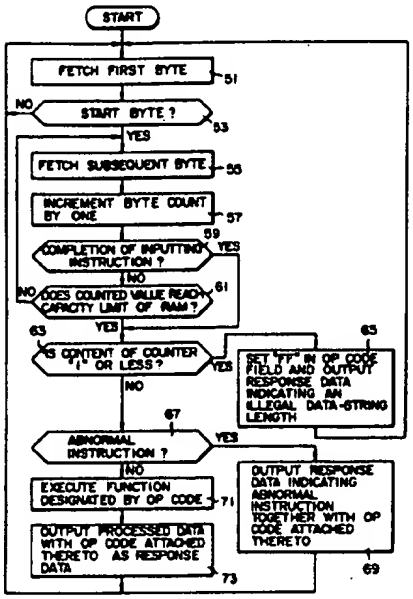
John D. Lenk, "The Protocol Link", Handbook of Communications, (1984) pp. 292–295.

Primary Examiner—Thomas C. Lee
Assistant Examiner—Robert B. Harrell
Attorney, Agent, or Firm—Cushman, Darby & Cushman

[57] **ABSTRACT**

According to a portable electronic apparatus processing system of this invention, when an IC card receives instruction data, it checks if a function code can be completely input. If the function code is not completely input, the IC card sets data "FF" (hexadecimal code) as the function code, and outputs it to an IC card reader/writer. When the IC card reader/writer outputs an instruction to the IC card, it appends sequential number data to the instruction. The IC card executes processing corresponding to the supplied instruction. When the IC card outputs the processed result to the IC card reader/writer, it appends the sequential number data from the IC card reader/writer to the processed result and outputs it.

6 Claims, 10 Drawing Sheets



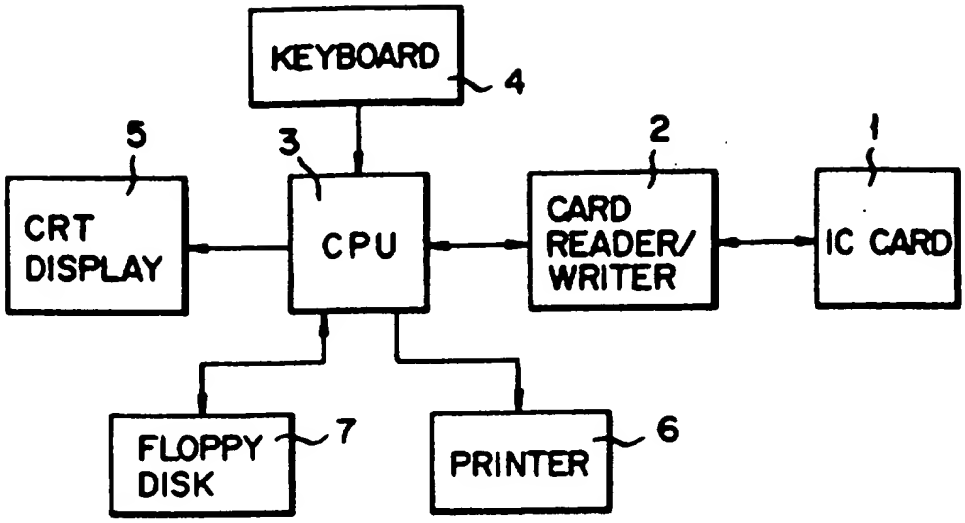


FIG. 1 (PRIOR ART)

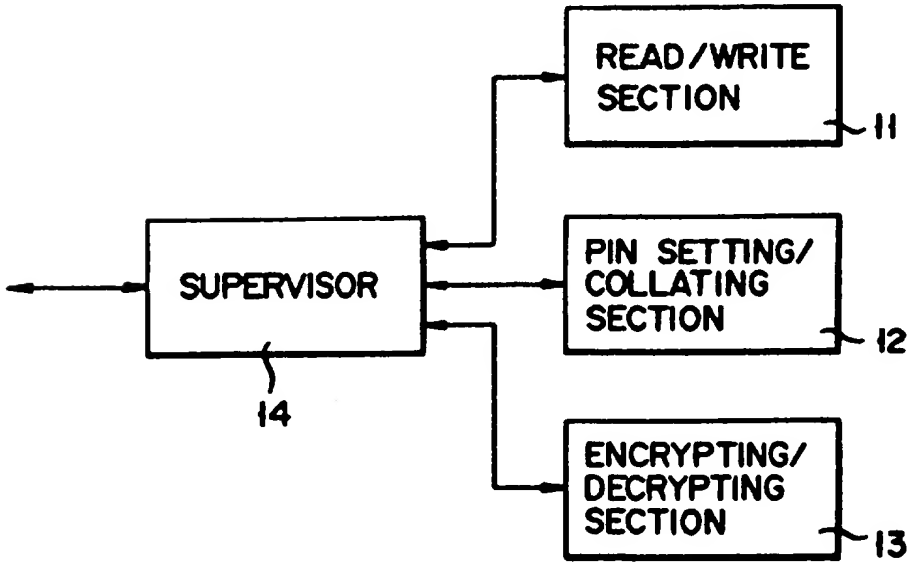


FIG. 2 (PRIOR ART)

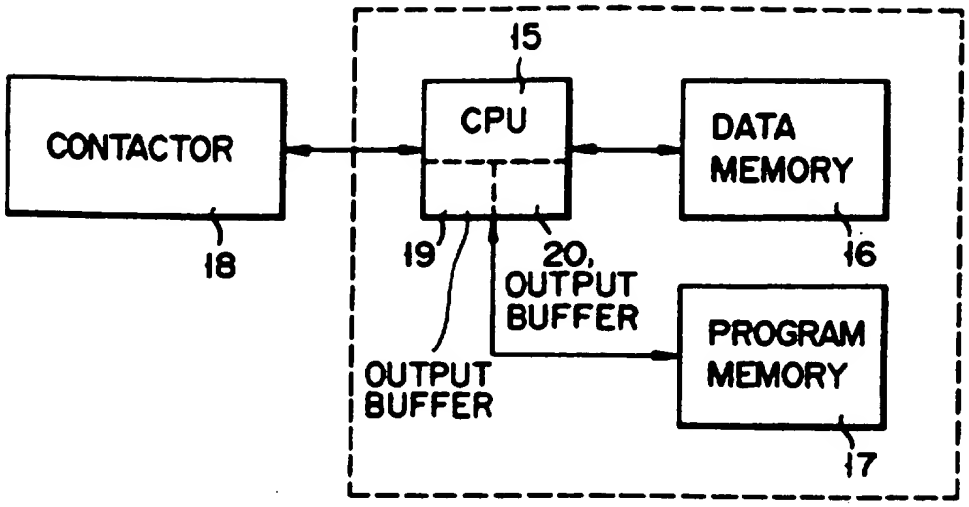


FIG. 3

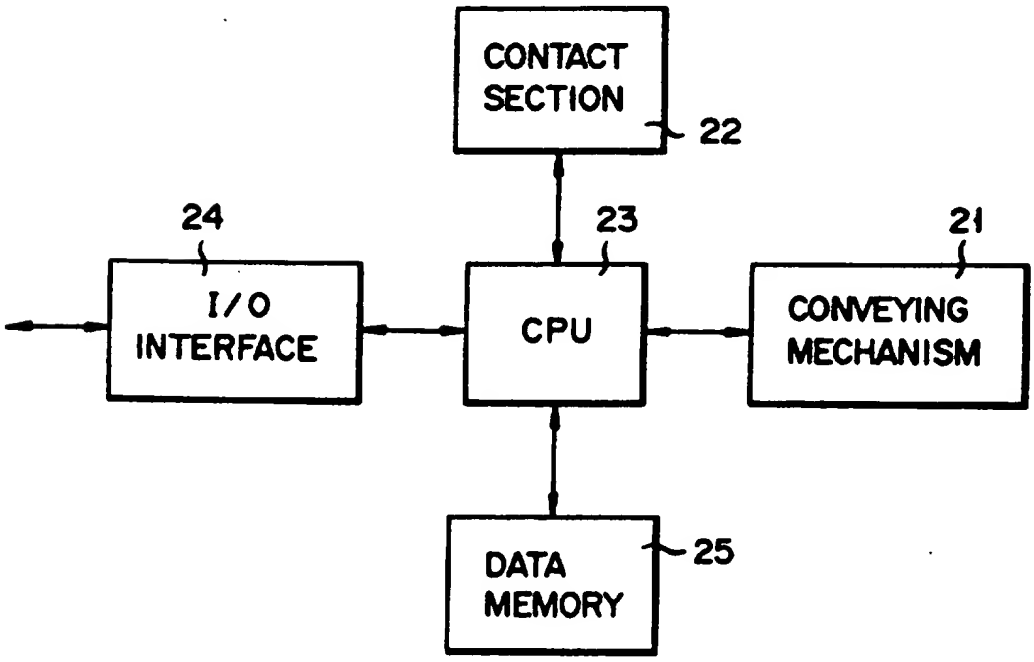


FIG. 4 (PRIOR ART)

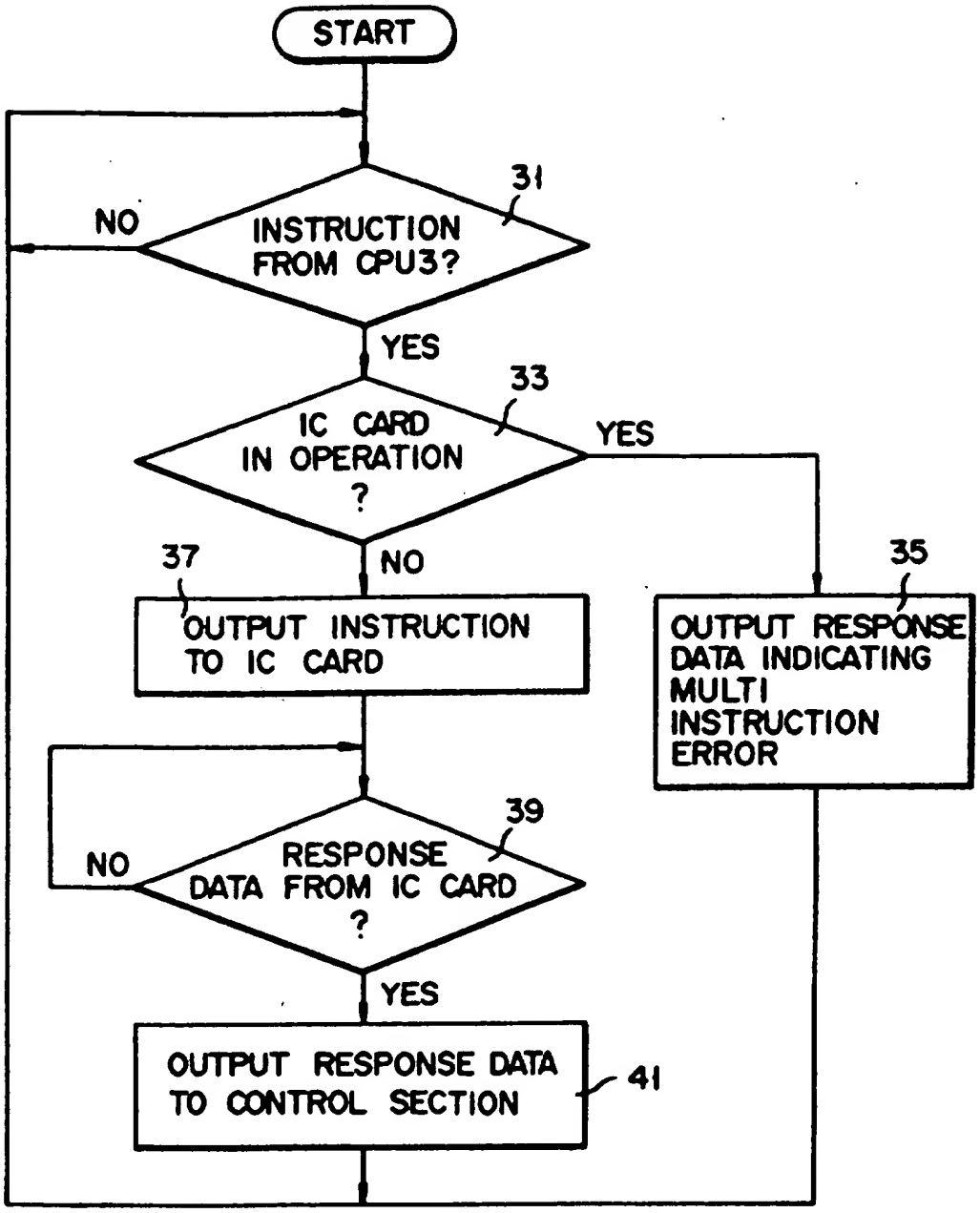
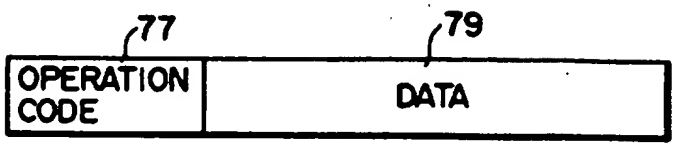


FIG. 5 (PRIOR ART)

FIG. 6A
(PRIOR ART)

FIG. 6B
(PRIOR ART)



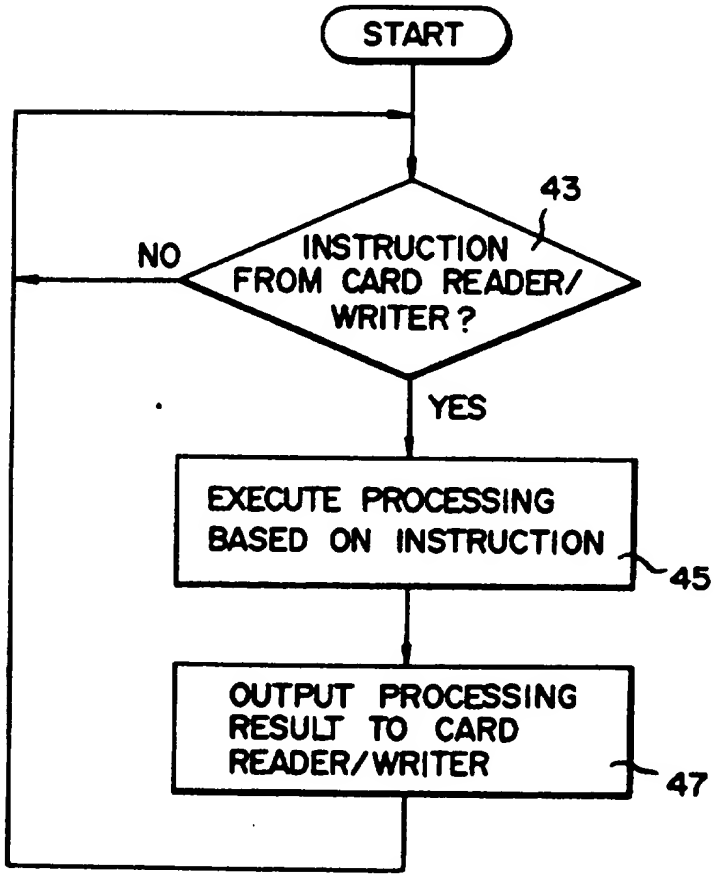


FIG. 7 (PRIOR ART)

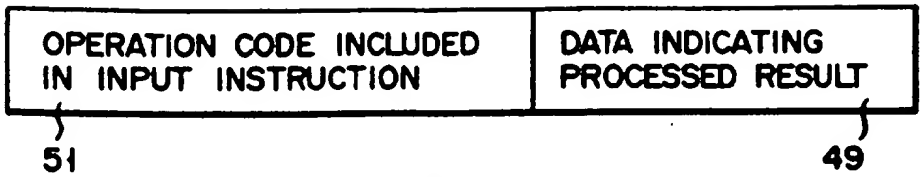


FIG. 8

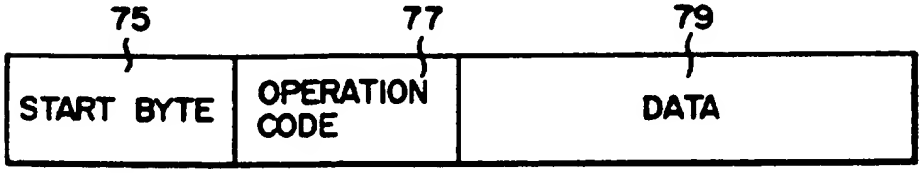


FIG. 10

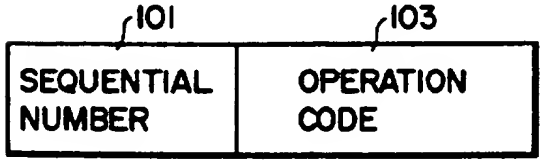


FIG. 11A

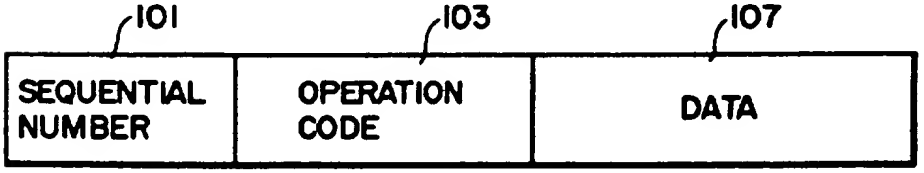


FIG. 11B

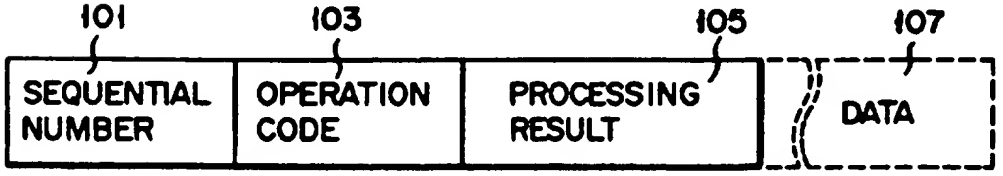


FIG. 13

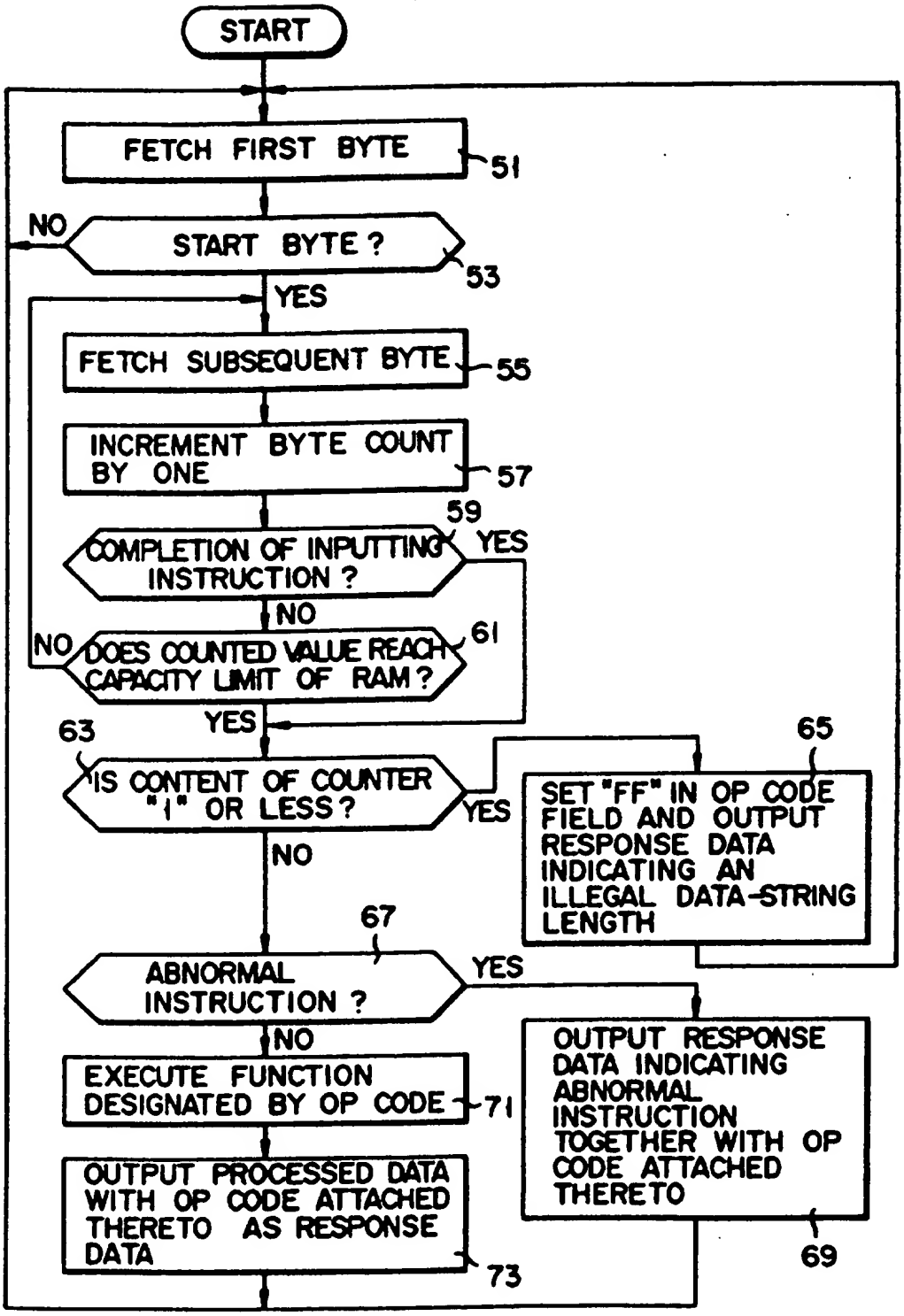
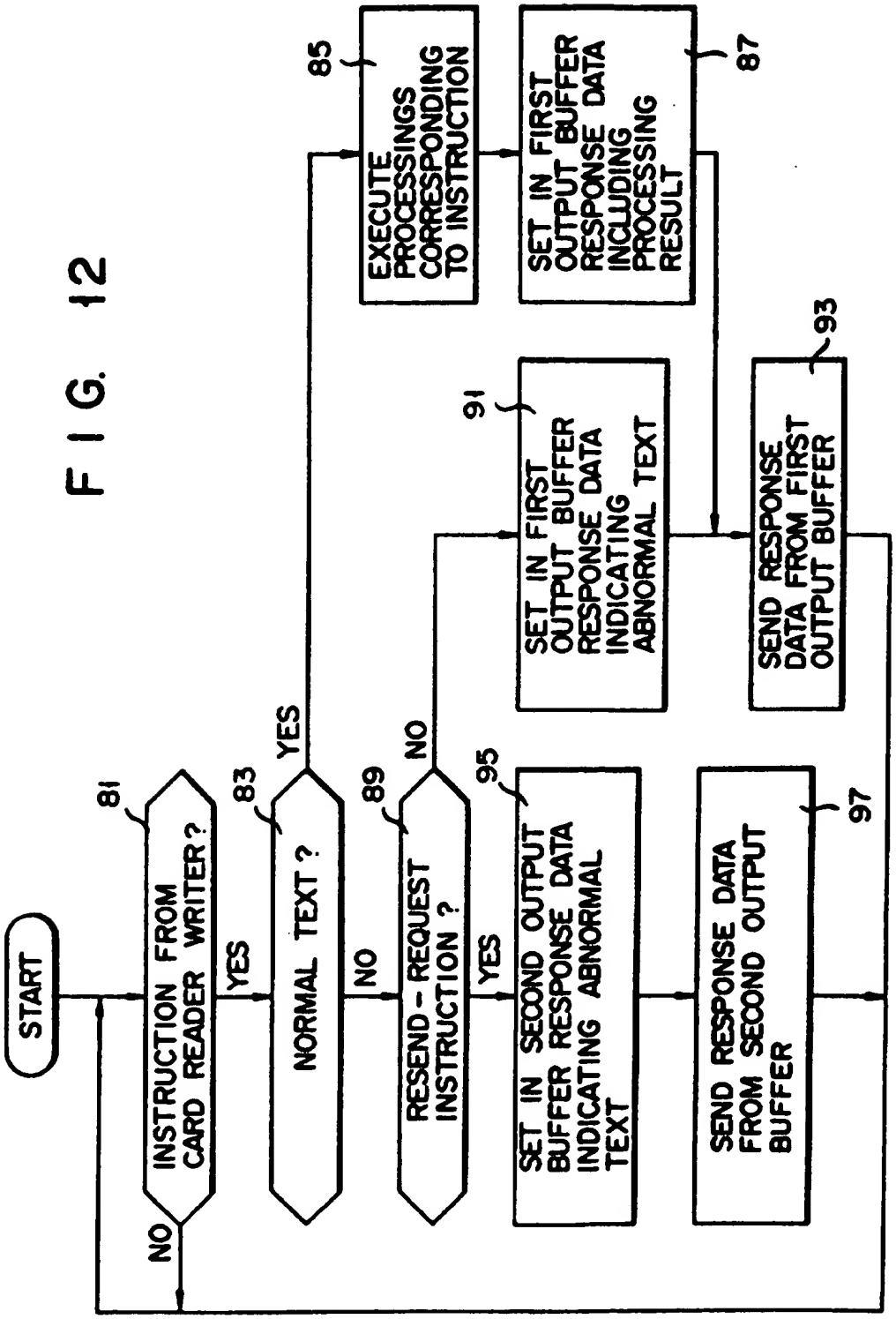

START
FETCH FIRST BYTE
51
NO
START BYTE ?
53
YES
FETCH SUBSEQUENT BYTE
55
INCREMENT BYTE COUNT BY ONE
57
59
COMPLETION OF INPUTTING INSTRUCTION ?
YES
NO
NO
DOES COUNTED VALUE REACH CAPACITY LIMIT OF RAM ?
61
YES
63
IS CONTENT OF COUNTER "1" OR LESS ?
YES
NO
65
SET "FF" IN OP CODE FIELD AND OUTPUT RESPONSE DATA INDICATING AN ILLEGAL DATA-STRING LENGTH
67
ABNORMAL INSTRUCTION ?
YES
NO
EXECUTE FUNCTION DESIGNATED BY OP CODE
71
OUTPUT PROCESSED DATA WITH OP CODE ATTACHED THERETO AS RESPONSE DATA
73
OUTPUT RESPONSE DATA INDICATING ABNORMAL INSTRUCTION TOGETHER WITH OP CODE ATTACHED THERETO
69

FIG. 9

FIG. 12


START
INSTRUCTION FROM CARD READER WRITER?
81
YES
NO
NORMAL TEXT ?
83
YES
NO
RESEND-REQUEST INSTRUCTION ?
89
NO
YES
85
EXECUTE PROCESSINGS CORRESPONDING TO INSTRUCTION
87
SET IN FIRST OUTPUT BUFFER RESPONSE DATA INCLUDING PROCESSING RESULT
91
SET IN FIRST OUTPUT BUFFER RESPONSE DATA INDICATING ABNORMAL TEXT
93
SEND RESPONSE DATA FROM FIRST OUTPUT BUFFER
95
SET IN SECOND OUTPUT BUFFER RESPONSE DATA INDICATING ABNORMAL TEXT
97
SEND RESPONSE DATA FROM SECOND OUTPUT BUFFER

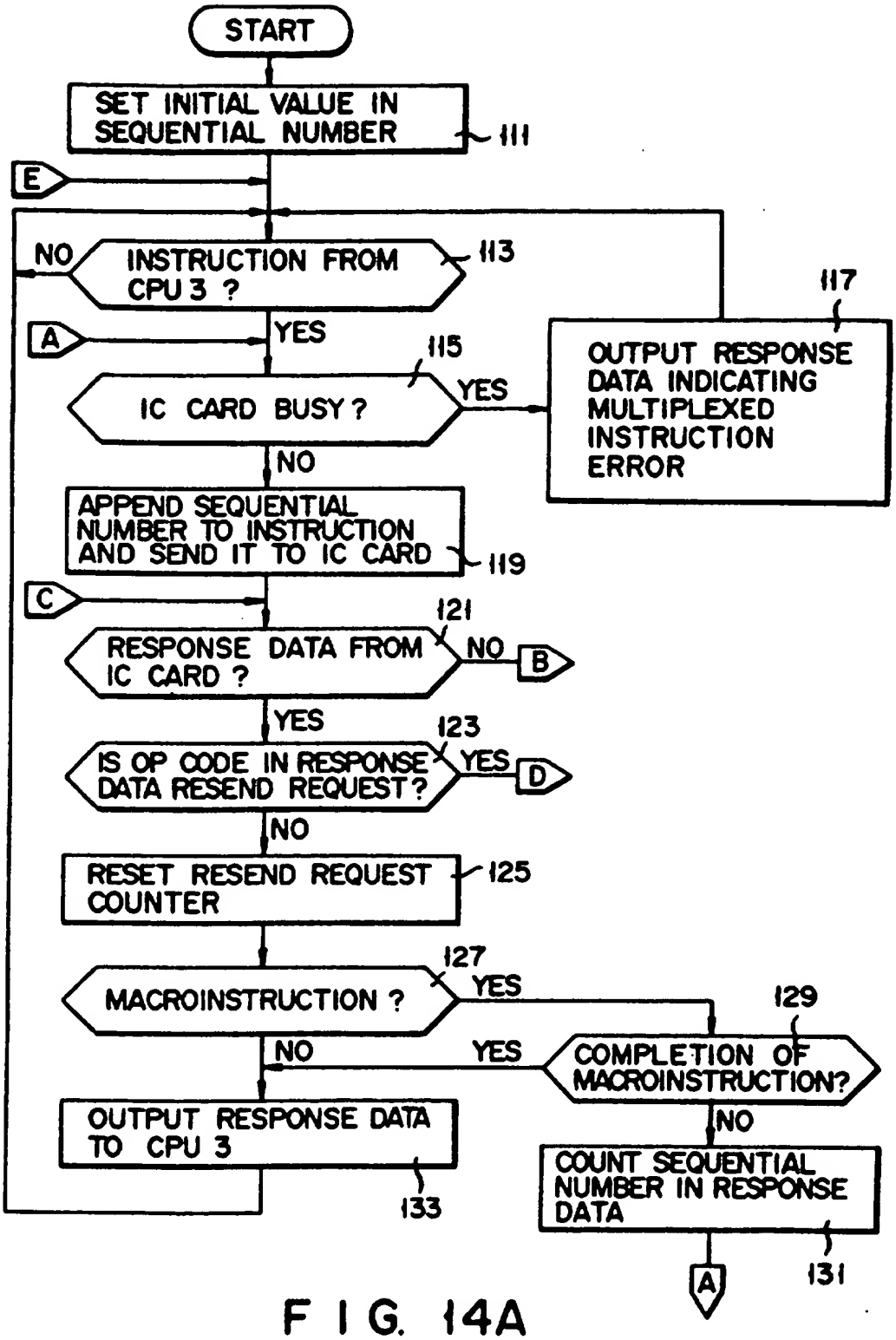

START
SET INITIAL VALUE IN SEQUENTIAL NUMBER
111
E
NO
INSTRUCTION FROM CPU 3 ?
113
A
YES
115
IC CARD BUSY ?
YES
117
OUTPUT RESPONSE DATA INDICATING MULTIPLEXED INSTRUCTION ERROR
NO
APPEND SEQUENTIAL NUMBER TO INSTRUCTION AND SEND IT TO IC CARD
119
C
121
RESPONSE DATA FROM IC CARD ?
NO
B
YES
123
IS OP CODE IN RESPONSE DATA RESEND REQUEST?
YES
D
NO
RESET RESEND REQUEST COUNTER
125
127
MACROINSTRUCTION ?
YES
129
NO
YES
COMPLETION OF MACROINSTRUCTION?
OUTPUT RESPONSE DATA TO CPU 3
133
NO
COUNT SEQUENTIAL NUMBER IN RESPONSE DATA
131
A

FIG. 14A

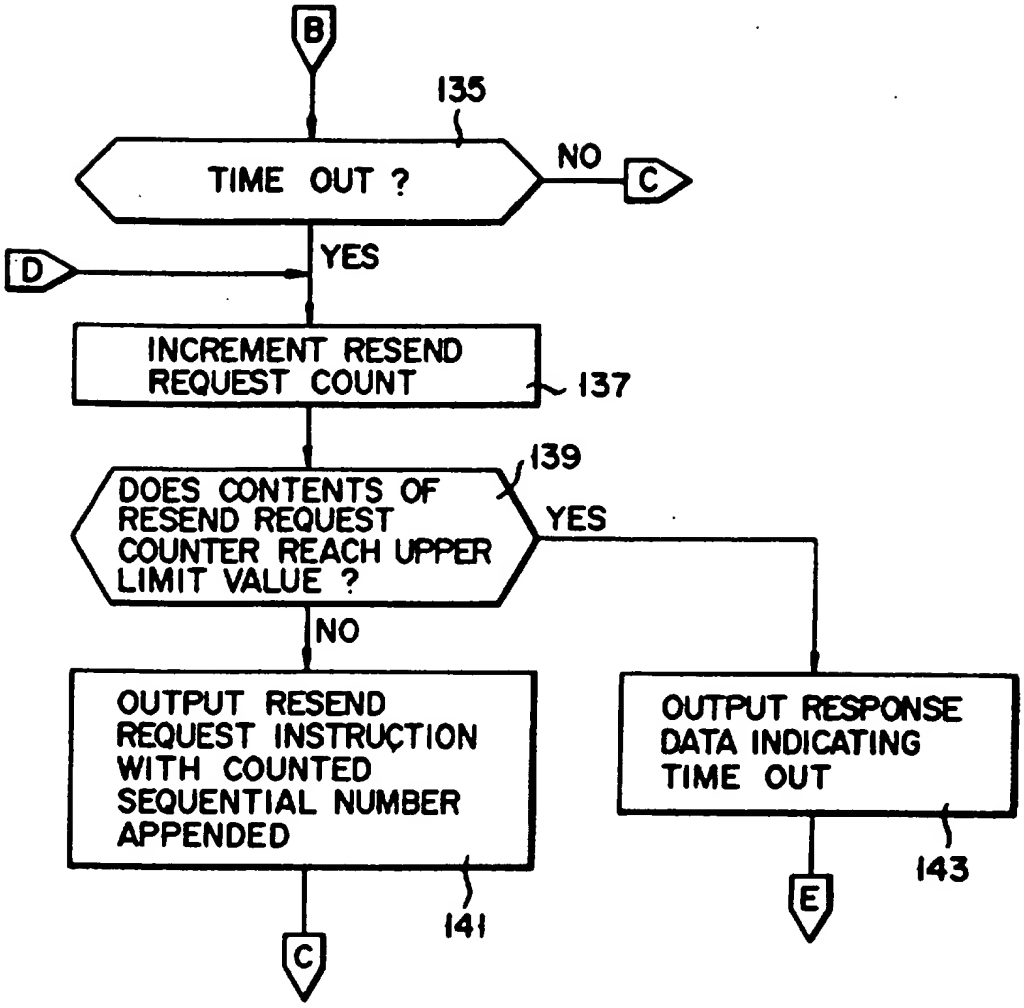
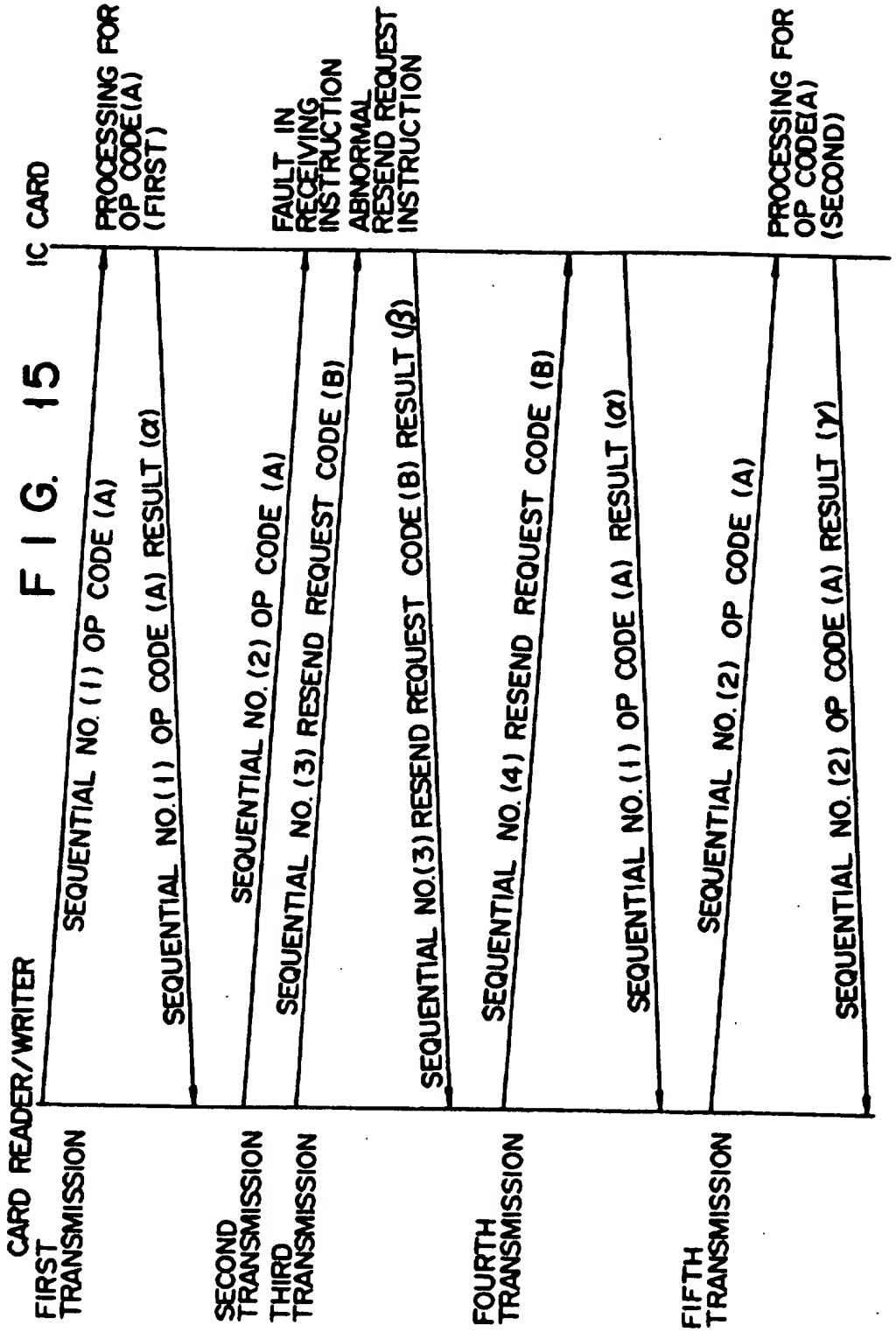

B
135
TIME OUT ?
NO
C
D
YES
INCREMENT RESEND REQUEST COUNT
137
139
DOES CONTENTS OF RESEND REQUEST COUNTER REACH UPPER LIMIT VALUE ?
YES
NO
OUTPUT RESEND REQUEST INSTRUCTION WITH COUNTED SEQUENTIAL NUMBER APPENDED
141
C
OUTPUT RESPONSE DATA INDICATING TIME OUT
143
E

FIG. 14B

FIG. 15


CARD READER/WRITER
IC CARD
FIRST TRANSMISSION
SEQUENTIAL NO.(1) OP CODE (A)
PROCESSING FOR OP CODE(A) (FIRST)
SEQUENTIAL NO.(1) OP CODE (A) RESULT (α)
SECOND TRANSMISSION
SEQUENTIAL NO.(2) OP CODE (A)
THIRD TRANSMISSION
SEQUENTIAL NO.(3) RESEND REQUEST CODE (B)
FAULT IN RECEIVING INSTRUCTION
ABNORMAL RESEND REQUEST INSTRUCTION
SEQUENTIAL NO.(3) RESEND REQUEST CODE (B) RESULT (β)
FOURTH TRANSMISSION
SEQUENTIAL NO.(4) RESEND REQUEST CODE (B)
SEQUENTIAL NO.(1) OP CODE (A) RESULT (α)
FIFTH TRANSMISSION
SEQUENTIAL NO.(2) OP CODE (A)
PROCESSING FOR OP CODE(A) (SECOND)
SEQUENTIAL NO.(2) OP CODE (A) RESULT (γ)

PROCESSING SYSTEM WHICH TRANSMITS A PREDETERMINED ERROR CODE UPON DETECTION OF AN INCORRECT TRANSMISSION CODE

This is a continuation of application Ser. No. 07/361,349, filed on Jun. 5, 1989, which was abandoned upon the filing hereof which is a continuation of Ser. No. 07/097,660 filed Sep. 16, 1987, now abandoned.

BACKGROUND OF THE INVENTION

The present invention relates to a processing system for a portable electronic apparatus, e.g., an IC card.

Recently, an IC card incorporating an IC chip having a nonvolatile data memory and a control element such as a CPU (Central Processing Unit) has been developed. An IC card of this type is normally operated using a card reader/writer. This operation is performed by an instruction supplied from the card reader/writer. The IC card decodes an operation code (OP code) of the received instruction, and executes a series of processing corresponding to a function indicated by the OP code. Then, the IC card outputs the processed result to the card reader/writer as response data. In this case, a function code included in the input instruction is attached to the response data to compensate for the irregular sequence of instructions exchanged between the IC card and the card reader/writer. More specifically, the OP code is extracted from the instruction stored in a buffer upon an input of the instruction data, and the extracted OP code is attached to the processed result when the processed result is output to the card reader/writer.

However, with this method, when no OP code is included in the instruction from the card reader/writer (in the case of an illegal instruction), or when the instruction cannot be normally received due to transmission noise, wrong data may be output to the card reader/writer as the OP code. This is because a portion corresponding to the OP code of the instruction stored in the buffer is simply extracted and attached to the processed result as the OP code. For this reason, when a sequence between the card reader/writer and the IC card is disordered, the compensation may be impossible. In particular, when instructions having the same OP code are successively transmitted a plurality of times, the sequence may often be disordered. When sequence disorder occurs upon transmission of data write instruction data, an abnormality such as double writing of data occurs.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a processing system for a portable electronic apparatus which can eliminate sequence disorder between a card reader/writer and an IC card when an illegal instruction is sent from the card reader/writer or when an instruction cannot be normally received due to transmission noise, and which can reliably prevent sequence disorder between the card reader/writer and the IC card when instructions having the same OP code are successively transmitted a plurality of times.

In order to achieve the above object, according to the present invention, there is provided a processing system constituted by a portable electronic apparatus handling apparatus for outputting an instruction having at least a function code field indicating a content of processing, and a portable electronic apparatus which is connected to the portable electronic apparatus handling apparatus as needed so as to receive the instruction supplied from the portable electronic apparatus handling apparatus, execute processing corresponding to the received instruction, and output a processed result with the function code attached thereto to the portable electronic apparatus handling apparatus, the improvement comprising:

checking means for checking if the function code of the instruction supplied from the portable electronic apparatus handling apparatus is input; and

output means for outputting, to the portable electronic apparatus handling apparatus, specific data indicating that the function code is not input when the checking means determines that the function code is not input.

According to the present invention, when IC card 1 fetches instruction data, it checks if a function code can be completely received. If it is not completely received, card 1 assigns data "FF" in an OP code portion of response data to be output to card reader/writer 2. As a result, wrong data cannot be output as an OP code, unlike in the conventional system. Therefore, even when illegal instruction data is sent from card reader/writer 2, or when an instruction cannot be normally received due to transmission noise, sequence disorder between card reader/writer 2 and IC card 1 can be eliminated.

A sequential number starting from a specific initial value is attached to instruction data output from the card reader/writer. The IC card receiving the instruction attaches the sequential number of the received instruction to response data and outputs it so that sequence disorder that cannot be prevented by response data with only a function code can be prevented. More specifically, since the sequential number attached to the instruction data is attached to the response data, the relationship between the instruction and the response data can be easily discriminated from the sequential number. In particular, when instructions having the same OP code are successively transmitted a plurality of times, sequence disorder between the card reader/writer and the IC card can be reliably prevented.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram showing the arrangement of a conventional card handling apparatus;

FIG. 2 is a block diagram showing functions of a conventional IC card;

FIG. 3 is a block diagram showing the arrangement of an IC chip incorporated in an IC card to which the present invention is applied;

FIG. 4 is a block diagram showing the arrangement of a conventional card reader/writer;

FIG. 5 is a flow chart for explaining the operation of a conventional card reader/writer;

FIGS. 6A and 6B show formats of a conventional instruction output from a card reader/writer;

FIG. 7 is a flow chart for explaining the operation of a conventional IC card;

FIG. 8 shows a format of response data output from an IC card;

FIG. 9 is a flow chart for explaining instruction fetching and processing operations of the IC card according to an embodiment of the present invention;

FIG. 10 shows a format of an instruction output from a card reader/writer;

FIGS. 11A and 11B show formats of an instruction output from the card reader/writer;

FIG. 12 is a flow chart for explaining the operation of the IC card in the embodiment of the present invention;

FIG. 13 shows a format of response data output from the IC card in the embodiment of the present invention;

FIGS. 14A and 14B are flow charts for explaining a detailed operation of the card reader/writer according to the embodiment of the present invention; and

FIG. 15 is a chart showing a detailed transmission sequence between the card reader/writer and the IC card in the embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

FIG. 1 shows the arrangement of a card handling apparatus used as a terminal for a home banking system or a shopping system to which an IC card as a portable electronic apparatus is applied. The card handling apparatus comprises card reader/writer 2 for performing read/write access of the IC card 1, keyboard 4 for inputting data such as a personal identification number (PIN), CRT display unit 5 for displaying transaction data, printer 6 for printing out the transaction data, floppy disk 7 for storing the transaction data, and central processing unit (CPU) 3 for controlling card reader/writer 2, keyboard 4, CRT display unit 5, printer 6, and floppy disk 7.

IC card 1 is carried by a user, subjected to collation with a PIN known only by a user upon purchasing of commodities, and used to store necessary data.

FIG. 2 shows functional blocks of IC card 1. IC card 1 comprises sections for executing basic functions, such as read/write section 11, PIN setting/collating section 12, and encrypting/decrypting section 13, and supervisor 14 for controlling these basic functions. Read/write section 11 performs data read, write, or erase access with data memory 16. PIN setting/collating section 12 performs storage and read inhibition processing of a PIN set by a user, and after the PIN is set, collates the set PIN to provide permission for the subsequent processing. Encrypting/decrypting section 13 performs encryption for preventing disclosure and forgery of communication data when data is transmitted from CPU 3 to another terminal through a communication line, and decrypts encrypted data. Data encryption is performed in accordance with an encryption algorithm having a sufficient encryption power, such as the Data Encryption Standard (DES). Supervisor 14 decodes an operation code (OP code) input from card reader/writer 2 or the OP code with data, and selects and executes a necessary function.

In order to effect these functions, IC card 1 is constituted by central processing unit (CPU) 15, non-volatile data memory 16 whose storage contents are erasable, program memory 17, and contactor 18 for electrically contacting card reader/writer 2. CPU 15, data memory 16, and program memory 17 comprise a commercially available microprocessor chip (e.g., Intel 8080 or an equivalent). CPU 15 has first output buffer 19 and second output buffer 20. Note that the number of output buffers is not limited to 2.

Program memory 17 comprises a mask read-only memory (ROM), and has a control program for controlling CPU 15. The control program includes a subroutine group for effecting a function corresponding to an OP code of an instruction supplied from card reader/writer 2. Data memory 16 comprises an electrically erasable programmable read-only memory (EEPROM).

Card reader/writer 2 exchanges an OP code or data with CPU 3 of IC card 1. More specifically, as shown in FIG. 4, card reader/writer 2 comprises conveying mechanism 21 for conveying IC card 1 inserted in a card insertion port (not shown) to a predetermined position, contact section 22 for electrically contacting contactor 18 of IC card 1 set at the predetermined position, CPU 23 for controlling the overall operation, I/O interface 24 adapted to cause CPUs 23 and 3 to exchange instruction and response data therebetween, and data memory 25 for storing data.

FIG. 5 is a flow chart showing the operation of card reader/writer 2. More specifically, it is checked in step 31 if an instruction is supplied from CPU 3. If NO in step 31, the flow returns to step 31, and CPU 23 awaits input data. If YES in step 31, it is checked in step 33 if IC card 1 is in operation. If YES in step 33, response data indicating a multi instruction error is output to CPU 3, and the flow returns to step 31.

If NO in step 33, control advances to step 37, and an instruction is output to IC card 1. In step 39, CPU 23 awaits response data from IC card 1. If the response data is detected in step 39, control advances to step 41, and the response data is output to CPU 3.

FIGS. 6A and 6B show instruction formats output to IC card 1. FIG. 6A shows an instruction format consisting of only an OP code 77, and FIG. 6B shows an instruction format consisting of an OP code 77 and data 79.

IC card 1 is operated in accordance with the flow chart shown in FIG. 7. CPU 15 awaits an input instruction from card reader/writer 2 in step 43. When CPU 15 detects the instruction from card reader/writer 2 in step 43, CPU 15 executes processing based on the instruction in step 45. Then, control advances to step 47, and the processing result is output to card reader/writer 2. The flow returns to step 43. FIG. 8 shows the format of response data in this case. As can be seen from FIG. 8, field 51 indicating a function code included in the input command is attached to data field 49 indicating the processed result.

An operation will now be described with reference to the flow chart in FIG. 9 wherein IC card 1 fetches an instruction and executes processing in accordance with the invention. When an instruction is to be sent from card reader/writer 2 to IC card 1, start byte field 75 is attached to an instruction in addition to OP code field 77 and data field 79, as shown in FIG. 10. In step 51, the first byte of the input instruction is fetched. It is checked in step 53 if the fetched byte is a start byte. If NO in step 53, the flow returns to step 51. If YES in step 53, the subsequent byte is fetched in step 55. In step 57, a byte count is incremented by one. It is then checked in step 59 if inputting of the instruction is completed. If YES in step 59, the flow advances to step 63. However, if NO in step 59, it is checked in step 61 if the byte count has reached a capacity limit of data memory 16. If NO in step 61, the flow returns to step 55, and the subsequent byte is fetched. On the other hand, if YES in step 61, the flow advances to step 63 to check if the count is "1" or less. When YES is obtained in step 61 and the flow advances to step 63, NO is always obtained in step 63. The capacity limit of the RAM in this case is 32 bytes, and the byte count has reached 32. Therefore, NO is always obtained in step 63.

When the flow advances from step 59 to step 63 and YES is obtained in step 63, although the input data string is completed, the byte count is "1" or less, and this means a substantial instruction has not been received and processing cannot be executed. This is because the OP code consists of 2 bytes. Therefore, if YES in step 63, data "FF" (hexadecimal code) is set in OP code field 77 in step 65, and response data indicating an illegal data-string length is output. More specifically, when CPU 15 receives an instruction from card reader/writer 2 and when it discriminates that no OP code is input, CPU 15 outputs, to card reader/writer 2, the processed result with specific data other than the OP code, e.g., a specific code "FF" (hexadecimal code) as response data.

If NO in step 63, it is checked in step 67 if an abnormal instruction such as parity abnormality is input. If YES in step 67, CPU 15 outputs response data indicating an abnormal instruction together with the OP code attached thereto, and the flow returns to step 51. However, if NO in step 67, CPU 15 executes a function designated by the OP code in step 71, and outputs, to card reader/writer 2, the processed result with the OP code attached thereto as response data in step 73.

In the above embodiment, data exchange between card reader/writer 2 and IC card 1 has been exemplified. The present invention can be applied to data exchange between CPU 3 and card reader/writer 2.

A second embodiment of the present invention will now be described. In this embodiment, in an instruction sent from card reader/writer 2 to IC card 1, a sequential number is attached to the instruction formats shown in FIGS. 6A and 6B, as shown in FIGS. 11A and 11B. A sequential number 10 is attached to an instruction output 103 from card reader/writer 2, and IC card 1 receiving the instruction attaches the sequential number of the received instruction to response data 107 and outputs it to card reader/writer 2.

The embodiment of the present invention will be described hereinafter with reference to the flow chart shown in FIG. 12. CPU 15 checks in step 81 if an instruction is sent from card reader/writer 2. If NO in step 81, CPU 15 awaits until the instruction arrives. If YES in step 81, it is checked in step 83 if the received text (instruction) is normal. More specifically, it is checked if an OP code included in the text corresponds to any of predetermined OP codes or if the text includes a parity error. If YES in step 83, CPU 15 executes processing corresponding to the input instruction in step 85, and sets the processed result in output buffer 19 in step 87. In step 93, CPU 15 then outputs response data including the processed result set in output buffer 19 to card reader/writer 2. Then, the flow returns to step 81, and CPU 15 awaits the next instruction from card reader/writer 2. The response data which is output from CPU 15 to card reader/writer 2 in step 93 consists of sequential number 101 attached to the instruction sent from card reader/writer 2, OP code 103 attached thereto, processing result 105, and data 107 which is attached if necessary, as shown in FIG. 13.

On the other hand, if NO in step 83, CPU 15 checks in step 89 if the instruction is a resend-request instruction. If NO in step 89, CPU 15 sets response data indicating an abnormal text in first output buffer 19 in step 91, and sends the response data set in first output buffer 19 to card reader/writer 2. Thereafter, the flow returns to step 81, and CPU 15 awaits the next instruction from card reader/writer 2. However, if YES in step 89, CPU 15 sets response data indicating an abnormal text in second output buffer 20 in step 95, and sends the response data set in buffer 20 to card reader/writer 2 in step 97. Thereafter, the flow returns to step 81.

In this embodiment, two output buffers are provided to IC card 1, and they are selectively used to store response data when a resend-request instruction is abnormal and to store data response data including the processed result. With this arrangement, response data including the processed result to be resent cannot be erased by setting response data indicating an abnormal text.

The operation of card reader/writer 2 will now be described with reference to the flow charts shown in FIGS. 14A and 14B. CPU 23 in card reader/writer 2 sets an initial value in a sequential number in step 111, and checks in step 113 if an instruction is supplied from CPU 3. If NO in step 113, CPU 23 waits until an instruction is supplied. If YES in step 113, CPU 23 checks in step 115 if IC card 1 is busy. If YES in step 115, CPU 23 outputs response data indicating a multiplexed instruction error to CPU 3 in step 117, and the flow returns to step 113. If NO in step 115, CPU 23 appends a sequential number to an instruction in step 119 and sends it to IC card 1. CPU 23 then checks in step 121 if response data is supplied from IC card 1. If NO in step 121, CPU 23 checks in step 135 if time is up (time out). If NO in step 135, the flow returns to step 121, and CPU 23 awaits response data from IC card 1. However, if YES is completed in step 129, CPU 23 outputs response data to CPU 3 in step 133. Thereafter, the flow returns to step 113, and awaits the instruction from CPU 3. If NO in step 127, CPU 23 outputs response data to CPU 3 in step 133. Then, the flow returns to step 113, and CPU 23 awaits the instruction from CPU 3.

FIG. 15 shows a detailed transmission sequence between card reader/writer 2 and IC card 1. In the first transmission, card reader/writer 2 sends an instruction with sequential No. (1) and OP code (A) to IC card 1. IC card 1 executes processing corresponding to OP code (A), and sends back response data including sequential No. (1), OP code (A), and processed result (α) to card reader/writer 2. Since the response data with sequential No. (1) is sent back from IC card 1, card reader/writer 2 sends an instruction with sequential No. (2) and OP code (A) to IC card 1 in the second transmission. Assume that IC card 1 fails to receive the instruction itself. In this case, for example, IC card 1 cannot receive the start byte. Card reader/writer 2 confirms that response data corresponding to the instruction when sequential No. (2) is not sent back from IC card 1. However, card reader/writer 2 cannot decide whether IC card 1 cannot receive the instruction or card reader/writer 2 cannot receive the instruction although the response data including the processed result is sent back from IC card 1. Therefore, card reader/writer 2 sends an instruction including sequential No. (3) and OP code (B) indicating a resend request to IC card 1. In the third transmission, assume that IC card 1 receives the instruction with sequential No. (3) from card reader/writer 2 but determines that the received text is abnormal (e.g., the OP code is undesirably converted to a nonregistered OP code during transmission). IC card 1 sends back response data including sequential No. (3), OP code (B) indicating the resend request, and processed result (β) indicating an abnormal text to card reader/writer 2. Upon reception of this data, card reader/writer 2 sends an instruction including sequential No. (4) and OP code

(B) indicating the resend request to IC card 1. IC card 1 detects the resend-request instruction with sequential No. (4). Since IC card 1 could not receive an instruction with sequential No. (2), the latest data set in output buffer 19 is the processed result corresponding to the instruction with sequential No. (1). Therefore, IC card 1 sends response data including sequential No. (1), OP code (A), and processed result (α) to card reader/writer 2. Since the processed result corresponding to the instruction with sequential No. (1) is sent back from IC card 1, card reader/writer 2 sends an instruction including sequential No. (2) and OP code (A) to IC card 1 in the fifth transmission. IC card 1 detects this instruction, and executes processing corresponding thereto. Then, IC card 1 sends back response data including sequential No. (2), OP code (A), and processed result (γ) to card reader/writer 2.

In the above embodiment, the IC card has been exemplified as a portable electronic device. However, the shape of the portable electronic device is not limited to a card-like shape. The hardware arrangement of the portable electronic device can be modified within the spirit and scope of the invention.

What is claimed is:

1. A data transmission system in which in IC card accepting device sends instruction data to an IC card, the instruction data including in an instruction code field which includes at least a function code in a function code field indicating a content of processing, and said IC card returning the function code in a received instruction data back to said IC card accepting device, wherein said IC card includes:

means for receiving the instruction data;

means for checking contents of said instruction code field to determine whether the function code is a preliminarily registered function code;

first returning means for setting the function code in the function code field and for returning the function code field to said IC card accepting device, only when said checking means indicates that the function code is the preliminarily registered function code; and

second returning means for setting data in the function code field indicating that the function code in the instruction data received by said receiving means is not the registered function code, and for returning the instruction code field to said IC card accepting device, only when said checking means indicates that the function code is not the registered function code and wherein said IC card accepting device includes:

means for receiving the data indicating that the function code is not sent by said second returning means; and

means for in response to the function code resending the instruction data to said IC card.

2. A system according to claim 1, wherein the data is data other than the function code, consists of the same number of bits as the number of bits of the function code, and is set in said function code field.

3. A system according to claim 1, wherein the specified data consists of all "1" bits, and the function code is an arbitrary bit string consisting of not all "1" bits.

4. A data transmission as in claim 1, wherein said checking contents means also includes means for counting a length of an input instruction data; means for determining if the input instruction data is completed; means for determining if the counted length of the input instruction data has reached a predetermined value, and means for setting said data in the instruction code field when said counted length does not reach said specified value when said input instruction data is completed.

5. A data transmission system in which an IC card accepting device sends instruction data to an IC card, the instruction data including at least a function code in a function code field indicating a content of processing, and said IC card returning the function code in a received instruction data back to said Ic card accepting device, wherein said IC card includes:

means for receiving the instruction data;

first determining means for determining whether or not the instruction data is received correctly by said receiving means;

second determining means for determining whether or not the function code included in the instruction data is a registered function code when said first determining means determines that the instruction data is received correctly by said receiving means;

first sending means for sending to said IC card accepting device data indicating that the instruction data is received incorrectly when said first determining means determines incorrect instruction of the instruction data;

second sending means for sending to said IC card accepting device data indicating that the function code included in the instruction data is not the registered function code when said second determining means determines that the function code in the instruction data is not the registered function code; and

third sending means for sending a processed result for the instruction data together with the function code to said IC card accepting device if said first determining means determines the correct reception of the instruction data and said second determining means determines that the function code in the correctly received instruction data is the registered function code.

6. The system according to claim 5, wherein said first determining means includes means for counting data length of the instruction data.

* * * * *